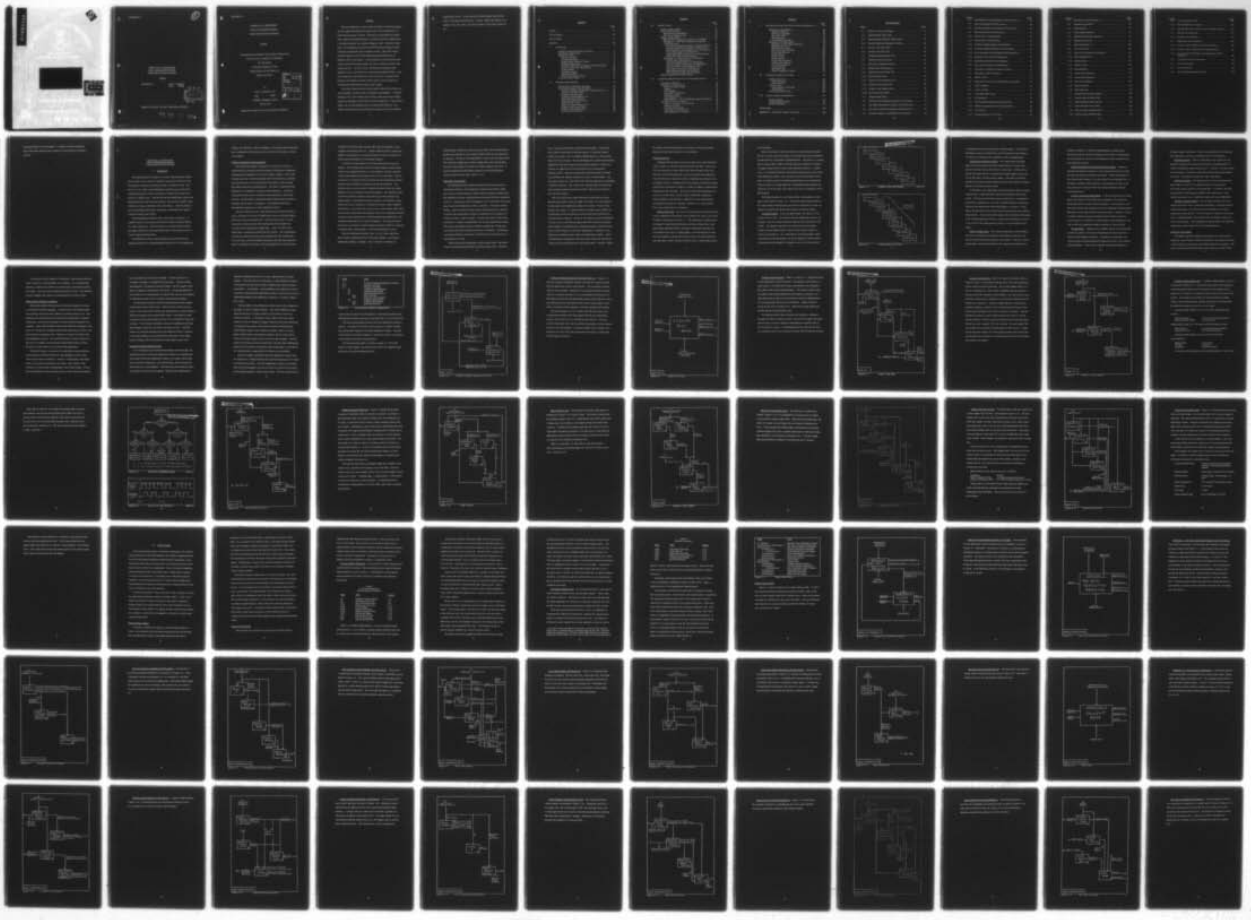
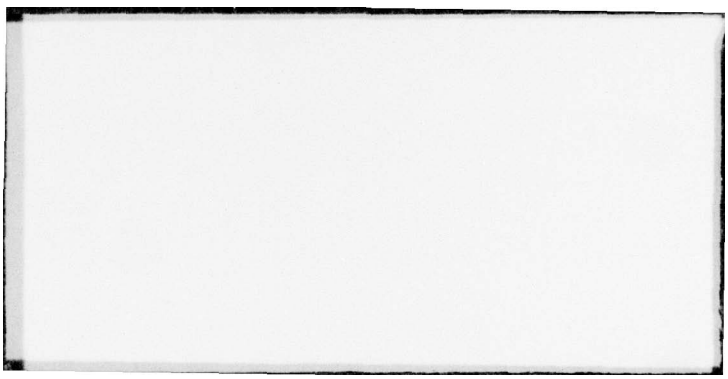
AD-A055 228 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN OF A LABORATORY DATA ACQUISITION SYSTEM (TIME DIGITIZATI--ETC(U)
MAR 78 J R MANEELY

UNCLASSIFIED AFIT/GCS/EE/78-4 NL

1 OF 3
AD
A055228





GCS/EE/78-4

(1)

DESIGN OF A LABORATORY DATA ACQUISITION SYSTEM (TIME DIGITIZATION SYSTEM)

THESIS

GCS/EE/78-4

John R. Maneely
Capt USAF

Approved for public release; distribution unlimited.

GCS/EE/78-4

DESIGN OF A LABORATORY DATA ACQUISITION SYSTEM (TIME DIGITIZATION SYSTEM)

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

John R. Maneely, B.S.

Capt USAF

Graduate Computer Science

March 1978

Preface

The time digitization system design provided an interesting opportunity to apply some high level design tools to the development of a small microprocessor system. The tools were primarily borrowed from software development theory, but they proved to be applicable to a combined hardware and software design as well. Of all the design tools, the techniques borrowed from the Structured Analysis Design Technique provided the most satisfying results. Since the Central Inertial Guidance Test Facility which sponsored the thesis is at Holloman AFB, New Mexico, it was possible to visit and discuss the system with laboratory personnel only once, and that was late in the design process. For that reason, the Structured Analysis style diagrams were very valuable for communicating difficult ideas. Even more important, the diagrams proved very easy to reorganize into hardware and software divisions which made a system design and especially the MSI hardware design quite simple.

This thesis would not have been possible without the assistance of several people, and their help is gratefully acknowledged. Lt Donald Pottenger who is the CIGTF project officer spent many hours finding answers to questions and learning to read SA diagrams. Captain Peter Miller was an understanding and encouraging thesis advisor, and Captain James Peterson provided the key design tools in his software

engineering course. Joyce Burnette's skillful typing improved the looks of this paper immeasurably. Finally, without the patience and support of my wife, Kris, this thesis would not have been possible at all.

Contents

		Page
Preface		ii
List of Figures		vii
List of Tables		xi
Abstract		xii
I.	Introduction	1
	Guidance Laboratory Data Acquisition	2
	Objectives of the Thesis	4
	Design Procedure	6
	System Life Cycle	6
	Conceptual Phase	7
	Requirements Definition Phase	9
	System Design Phase	9
	Hardware Selection and Software Structure Phase	10
	Circuit Layout and Coding Phase	10
	Testing Phase	10
	Integration Phase	11
	Operational Phase	11
	Iterations Between Phases	11
	Overview of the Thesis	11
II.	Requirements Definition	13
	Requirements Definition Language	14
	Structured Analysis Activity Model	15
	Perform Guidance Equipment Testing (Node A-1)	18
	Acquire Test Data (A-0)	19
	Acquire Test Data (A0)	20
	Condition Digital Signals (A1)	21
	Determine Net Count (A2)	22
	Identify Selected Events (A3)	25
	Skip N Events (A32)	26
	Delay for Time Window (A33)	27
	Prepare Data Record (A4)	28
	Prepare Data Sample (A44)	29
	Send Data Record (A45)	30

Contents

		Page
III.	System Design	32
	System Design Method	32
	Concurrent Activities	33
	Special Purpose Hardware	34
	Processor Requirements	36
	System Design Model	38
	Operate Time Digitization System (A-1/Design)	39
	Subsystem 1: Run Data Acquisition Programs (A-0/Processor)	40
	Run Data Acquisition Programs (A0/Processor)	41
	Process Program Commands (A1/Processor)	42
	Send Hardware Control Signals (A12/Processor)	43
	Form Data Sample (A2/Processor)	44
	Select Best Option Parameters (A3/Processor)	45
	Send Data Record (A4/Processor)	46
	Subsystem 2: Collect Data (A-0/Hardware)	47
	Collect Data (A0/Hardware)	48
	Condition Digital Signals (A1/Hardware)	49
	Control Hardware Operation (A2/Hardware)	50
	Store Hardware Control Signals (A21)	51
	Determine Net Count (A3/Hardware)	52
	Signal Selected Event (A4/Hardware)	53
	Save Data for Sample (A5/Hardware)	54
	System Design Phase Observations	55
IV.	Hardware Selection and Circuit Layout (Subsystem 1)	57
	Processor Selection	59
	Subsystem 1 Organization	65
	Basic Processor Group	65
	Memory	65
	I/O Decoders	67
	Subsystem 2 Hardware	67
	Interrupt Control	67
	Minicomputer I/O	68
	Minicomputer - Time Digitization System Interface	68
	Subsystem 1 Circuit Layout	70
	Basic Processor Group Circuits	70
	Memory Circuits	73
	I/O Decoder Circuits	73
	Interrupt Control Circuits	77
	Minicomputer I/O Circuits	79

Contents

		Page
V.	Hardware Selection and Circuit Layout (Subsystem 2) . . .	86
	Subsystem 2 Organization	88
	Signal Conditioner.	88
	Hardware Controller	88
	Net Count Logic.	91
	Event Selector	92
	Data Sample Registers	94
	Subsystem 2 Circuit Layout	94
	Hardware Control Signal Registers	96
	Elapsed Time Clock	96
	Input Synchronization	99
	Net Count Logic Circuits	100
	Pulse Edge Selector	106
	Event Counter	106
	Window Timer	108
	Event Signaler	111
	Event Time Registers	111
	Analog Data Registers	113
	Net Count Registers	115
	Status Register	115
	Interrupt Logic	117
	Signal Propagation Delays	117
VI.	Software Structure and Coding	119
	Software Structure	121
	System Start-up	122
	Data Collection	126
	Data Formats.	134
	Minicomputer Commands	134
	Data Samples	136
	Flow Charts and Coding	139
VII.	Results and Recommendations	151
	Design Results	151
	Design Method Results.	154
	Recommendations	156
Bibliography	. . . . .	159
Appendix A:	Structured Analysis Diagrams	160

List of Figures

Figure		Page
1-1	Software Life Cycle Phases	8
1-2	Digital System Life Cycle	8
2-1	Requirements Definition Model Index	17
2-2	Perform Guidance Equipment Testing	18
2-3	Acquire Test Data (A-0)	19
2-4	Acquire Test Data (A0)	20
2-5	Condition Digital Signals (A1)	21
2-6	Determine Net Count (A2)	22
2-7	Net Count Computation Rules	24
2-8	Sample Net Count Sequence	24
2-9	Identify Selected Events (A3)	25
2-10	Skip N Events (A32)	26
2-11	Delay for Time Window (A33)	27
2-12	Prepare Data Record (A4)	28
2-13	Prepare Data Sample (A44)	29
2-14	Send Data Record (A45)	30
3-1	Design Model Index	38
3-2	Operate Time Digitization System (A-1/Design)	39
3-3	Run Data Acquisition Programs (A-0/Processor)	40
3-4	Run Data Acquisition Programs (A0/Processor)	41
3-5	Process Program Commands (A1/Processor)	42

Figure		Page
3-6	Send Hardware Control Signals (A12/Processor)	43
3-7	Form Data Sample (A2/Processor)	44
3-8	Select Best Option Parameters (A3/Processor)	45
3-9	Send Data Record (A4/Processor)	46
3-10	Collect Data (A-0/Hardware)	47
3-11	Collect Data (A0/Hardware)	48
3-12	Condition Digital Signals (A1/Hardware)	49
3-13	Control Hardware Operation (A2/Hardware)	50
3-14	Store Hardware Control Signals (A21/Hardware)	51
3-15	Determine Net Count (A3/Hardware)	52
3-16	Signal Selected Event (A4/Hardware)	53
3-17	Save Data for Sample (A5/Hardware)	54
4-1	Hardware and Software Design Activities	58
4-2	Subsystem 1 Block Diagram	66
4-3	8080A-1 CPU	71
4-4	8224 Clock Generator and 8228 System Controller . . .	72
4-5	3628-4 PROMs	74
4-6	2111A-2 RAMs	75
4-7	RAM Chip Select Logic	75
4-8	I/O Decoders	76
4-9	8259 Programmable Interrupt Controller	78
4-10	8255A Programmable Interrupt Controller	80
4-11	I/O Timers	81
4-12	Timing Chart for I/O Timers	84

Figure		Page
5-1	Subsystem 2 Block Diagram	89
5-2	Hardware Controller	90
5-3	Event Selector	93
5-4	Data Sample Registers	95
5-5	Hardware Controller Registers	97
5-6	Elapsed Time Clock	98
5-7	Input Synchronizer	100
5-8	Net Count Logic	101
5-9	Net Count Overflow and Underflow	107
5-10	Pulse Edge Selector	107
5-11	Event Counter	107
5-12	Window Timer	109
5-13	Event Signaler	112
5-14	Event Time Registers	112
5-15	Analog Data Registers	114
5-16	Net Count Registers	116
5-17	Status Register	116
5-18	Interrupt Logic	116
6-1	System Start-up State Diagram	123
6-2	System Start-up State Table	125
6-3	Data Collection State Diagram	127
6-4	Data Collection State Table	128
6-5a	Minicomputer Command Word	135
6-5b	Nominal Time Window Value	135

Figure		Page
6-5c	I/O Coordination Word	135
6-6a	Data Sample Byte 1 Format	138
6-6b	Data Sample Byte 10 Event Time and Status Format . .	138
6-7	Process ZR Flowchart	141
6-8	Process ZA7 Flowchart	143
6-9	Subprocess ZA71 Flowchart	144
6-10	Process ZAA2 (EVENT Interrupt) Flowchart	145
6-11	Process ZAA1 (ANALOG Interrupt) Flowchart	147
6-12	Process ZAD7 (CONTROL) and Subprocess ZAD73 Flowchart.	148
A-1	Top-down View of an SA Model	162
A-2	Arrow Definitions	162
A-3	Arrow Branches	164
A-4	Arrows Showing Mutual Control	164

List of Tables

Table		Page
I	Functions Requiring Special Purpose Hardware . . .	34
II	Software Functions	37
III	8 Bit Microprocessors Considered for the Time Digitization System	63

Abstract

A design was developed to show the feasibility of a special microprocessor based data acquisition system called a time digitization system, which is to be used during tests of inertial guidance components for Air Force weapon systems. The time digitization system accepts two channels of digital pulse inputs and up to eight channels of analog inputs. A record, called a net count, of changes in the phase relationship between pulses on the two digital channel inputs is maintained, and at the occurrence of designated pulses on one of the digital channels, a time value, analog data, and the net count are saved. The acquired data is organized into records and transmitted to a minicomputer.

A digital system life cycle was developed to serve as a framework for the design project. Within the life cycle, requirements definition, system design, hardware selection/software structure, and circuit layout phases were completed. A technique patterned after Structured Analysis was used to construct a requirements definition model. The requirements model was converted to a system design model by separating hardware and software functions. An Intel 8080 microprocessor system was selected to perform the software functions, and MSI circuits were selected to perform special purpose hardware functions which were beyond the capability of a microprocessor. Circuit layouts for both the 8080 microprocessor system and the special

purpose hardware were developed. A software structure patterned after finite state automata was created to control the data acquisition process.

DESIGN OF A LABORATORY DATA ACQUISITION SYSTEM (TIME DIGITIZATION SYSTEM)

I Introduction

This paper presents a design for a small, special purpose digital device which is to be used for acquisition of precision data from inertial guidance instruments undergoing static and dynamic tests. The device is to be called a time digitization system because all the data acquired during tests must be related to an elapsed time value maintained by a digital clock. The idea for the time digitization system was developed by the Central Inertial Guidance Test Facility (CIGTF) of the 6585th Test Group at Holloman AFB, New Mexico. The time digitization system design described in this paper is developed from specifications provided by the CIGTF.

The development procedure used in this paper is top-down. A modified version of the Structured Analysis Design Technique (SADT) is a major design tool. Since the design is a masters thesis project, this paper emphasizes the intermediate steps of the design process as much as the final design itself.

The following sections of the Introduction provide background material necessary for understanding the function of a time digitization

system, the objectives of this investigation, the general design approach that is employed in this project, and an overview of the topics covered in this paper.

Guidance Laboratory Data Acquisition

The 6585th Test Group's Guidance Test Division or CIGTF is responsible for evaluating inertial navigation and guidance components and subsystems designed for Air Force weapon systems. Part of the evaluation process involves operating the inertial hardware, exposing the test specimen to a precisely known environment, and collecting data which represents its performance. The data is analyzed and the performance is modeled to evaluate the test specimen. Laboratory instruments which are now used make data acquisition a multi-step process, and that leads to undesirable delays between the time data is collected and the time it is analyzed. The equipment deficiency prompted the proposal for a time digitization system.

During a laboratory test, output data from the inertial test specimen is produced in the form of digital pulse trains and analog signals which are stored on magnetic tape. Once the tape is full, it can be removed to another location where appropriate data samples are extracted and stored again on digital tape. Later, the data on the digital tape is analyzed to produce the test results. The computations necessary to extract the proper data samples from the original tape are not trivial, and if the standard laboratory HP 2100 class minicomputer is employed for the task, little computation time is left for data

validation and interim data analysis which the minicomputer is also capable of performing (Ref 1:1). Another difficulty which complicates software requirements for final data analysis is that no standard format has been developed for storing data samples.

A number of improvements in the data acquisition process are desired. If the digital pulse trains and analog signals created during a test can be supplied directly to a minicomputer, the time required for the intermediate data storage process can be eliminated. What is necessary is an interface device between the test instruments monitoring the inertial navigation component and the minicomputer. The interface device must be able to translate the pulse trains and analog signals into a form useable by the minicomputer. By making the interface device powerful enough to select the proper data samples and prepare them in a standard format for storage, minicomputer computation time can be released for a significant amount of data validation and analysis. Since tests may run as long as a few months, data validation and some preliminary data analysis could save a substantial amount of time by early identification of malfunctioning test equipment or substandard performance by an inertial component. The proposed interface device, in essence a special purpose processor which is to operate in parallel with a minicomputer, is the time digitization system.

The time digitization system, then, is to be a data acquisition device which accepts pulse trains and analog signals, and produces digital data samples as outputs. Since a number of methods for

selecting data samples are used at various times, the time digitization system must operate in any of several modes which can be selected by an operator. The term "time digitization" comes from the requirement that each data sample must contain a digital time value representing the time of the sample in relation to a master elapsed time clock. A final and significant requirement imposed on the time digitization system is that it be low cost. (Ref 1:1, 4-5)

Objectives of the Thesis

The Statement of Work which the Guidance Division produced to specify the functions a time digitization system must perform states that a low cost microcomputer system should be capable of performing the repetitive and time consuming tasks associated with data acquisition (Ref 1:1). The purpose of this investigation is to verify that a microcomputer-based time digitization system is possible. During the system design, costs incurred by various system specifications are analyzed to provide the information necessary to determine if any changes are required in basic system concepts. Because the results of this study may lead to a change in specifications for the time digitization system, the scope of the design project is limited to preparing a design document which shows the necessary hardware and software. Purchasing and testing hardware components is deferred until a final decision on specifications is made.

Aside from the time digitization system design itself, this thesis is also concerned with the effects of several design tools. The first

tool is a general framework to guide the design effort. This framework, which is described in the next section, is called the Digital System Life Cycle, and is a slightly modified form of a common Software Life Cycle concept. The second design tool consists of procedures adapted from the Structured Analysis Design Technique. These procedures are applied first to the development of a requirements definition model. Then the requirements definition model is divided into hardware and software portions to create a system design. The last major design tool is the finite state automata approach to developing a software structure. The reasons for selecting SADT and the finite state automata software structure and their effects are described in the body of this paper.

Since microprocessor based laboratory instruments currently perform many types of data acquisition (Ref 2:65), it is reasonable to assume that a time digitization system can be built. Using similar systems as a guide, it is expected that a microprocessor system can be dedicated to data acquisition and need not have the flexibility of a general purpose computer. However, the system must be controllable at least as far as selecting the appropriate data acquisition program. Some method for modifying data acquisition programs should be available so future changes in laboratory equipment or techniques do not destroy the usefulness of the entire time digitization system. These few preliminary assumptions about the nature of the time digitization system form the starting point for the design effort. However, before

the results of the design project are presented, the general design approach of this thesis should be clearly defined.

Design Procedure

Designing with microprocessors and other large scale integrated (LSI) circuits is a new and relatively unstructured field. Only since 1975 has much been written which deals with the subject (Refs 3;4), and most of the writing has been in journal articles about applying a class of devices to a specific problem or applying one product to several types of tasks. Perhaps because the variety of LSI devices being produced has expanded so rapidly, no widely accepted, comprehensive design theory has emerged. The result is that most LSI designs are done on an ad hoc basis, and the choice of hardware components usually depends on the designer's experience or the programming development aids that are available. However, on a more general level some design theory is available.

System Life Cycle. One way to view the process of design development is suggested by the phases of the software life cycle, one version of which is presented in Figure 1-1. The progression of phases from conceptual to requirements definition and then to design show a top-down design approach which is generally considered most efficient. The software life cycle is usually applied to the development of software for hardware systems that already exist or at least which have been specified, but the concept of the life cycle is sufficiently general

to be extended.

When considering all the tasks which can be performed by digital devices, a broad range of functions can be found which lend themselves to both hardware and software implementations. Therefore it is reasonable to assume that a high level design approach suitable for software can apply equally well to at least some hardware functions. Furthermore, in a combined hardware and software design project such as the time digitization system, many functions cannot be allocated to hardware or software until more detailed specifications can be developed. What is needed is a design approach encompassing both hardware and software, and in this paper the software life cycle is modified to serve that purpose.

The digital system life cycle that provides a framework for this paper is presented in Figure 1-2. The actual system design begins in the requirements definition phase and ends at integration, but the conceptual and operational phases both influence the system design.

Conceptual Phase. In the conceptual phase, the idea for a new system comes to life. The idea for the time digitization system, for example, came as a likely method to improve data acquisition in the CIGTF. This phase is the province of those who will eventually benefit from the digital system, and not of the system designer in normal circumstances. As a consequence, the documentation which represents the conceptual phase can not be expected to define system requirements well enough for a designer's use, but it does provide

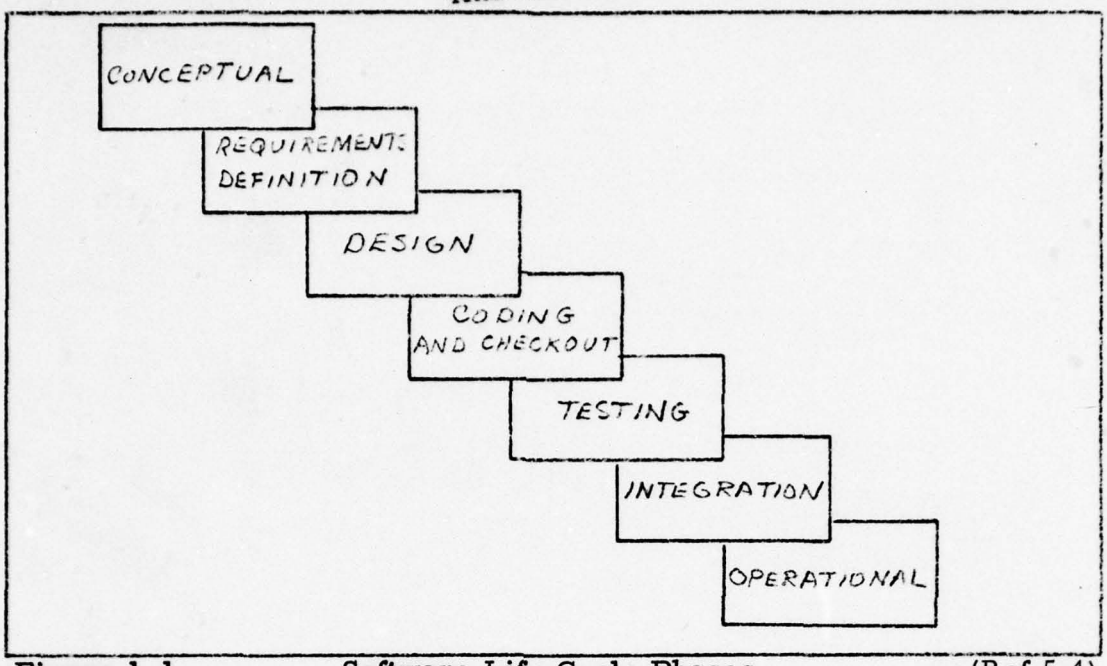


Figure 1-1 **Software Life Cycle Phases** (Ref 5:4)

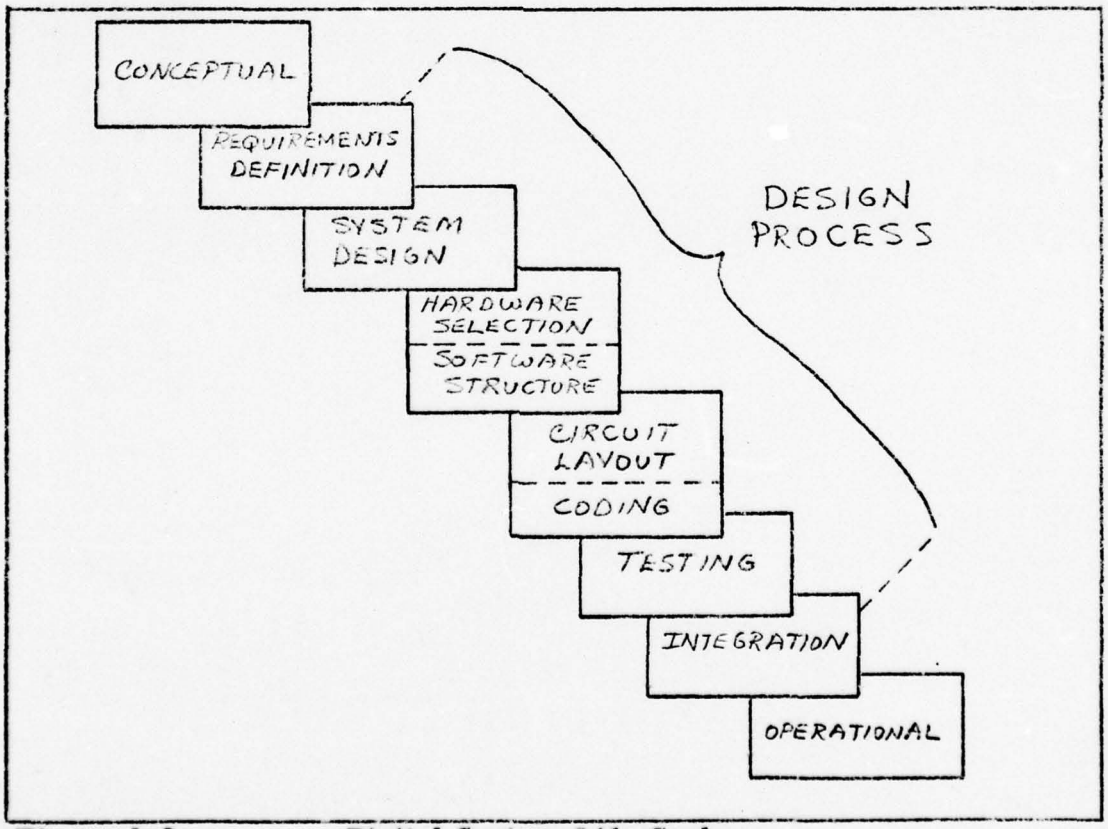


Figure 1-2 **Digital System Life Cycle**

a starting point for the requirements definition phase. For the time digitization system, the Statement of Work produced by the Guidance Test Division constitutes the completion of the conceptual phase.

Requirements Definition Phase. The requirements definition phase consists of specifying exactly what functions a system must perform and what timing restrictions it must meet. At this level of design, the things that are of interest are system inputs, the processing of the inputs, and the system outputs. The choice of specific hardware and coding methods should be left for later phases so design options are not prematurely limited.

In this thesis, the methodology and documentation chosen to define requirements is patterned after a Structured Analysis (SA) activity model. The Structured Analysis Design Technique was developed by the SofTech corporation as a precise, graphic method for identifying functions and showing their interrelationships in a system. In this paper, the SA model is also used to specify performance and interface requirements. Because the model repeats all the pertinent information in the original Statement of Work, and because the Statement of Work is not a part of the design process proper, it is not included in this paper.

System Design Phase. The system design phase of the software life cycle is divided into two phases in the digital system life cycle because both hardware and software are involved. In the new system design phase, functions identified in the requirements definition are

assigned hardware or software implementations as appropriate. Structured Analysis modules lend themselves to easy rearrangement, and that method was chosen for developing the system design for the time digitization system.

Hardware Selection and Software Structure Phase. Hardware selection and software structure are two separate but closely related functions forming the fourth life cycle phase. Since LSI hardware is less flexible than software, it is more likely to be developed first. In this paper, hardware block diagrams are prepared, and then a software structure is designed which can operate efficiently with the selected hardware.

Circuit Layout and Coding Phase. In the circuit layout and coding phase, the design work becomes more mechanical since the major design decisions are made in previous phases. Circuit layout is considered to include logic equations and circuit diagrams showing specific integrated circuit chips and their interconnections. Coding, of course, is simply writing the programs defined by the software structure. Circuit layout for virtually all of the hardware is presented here, and code for the most important software functions is also provided.

Testing Phase. Although some validation should occur along with the previous phases, during the testing phase the entire system is checked as thoroughly as possible. Test plans are advisable, and results should be checked against the approved version of the requirements definition. Tests can be conducted with actual hardware or

through computer simulation. Some recommendations for testing the time digitization system are included at the end of this paper.

Integration Phase. Plans for integrating a new system into its intended operating environment begin with the conceptual phase and continue through the entire design process. However, once the system is constructed, its interfaces with other systems must be thoroughly exercised. This phase is beyond the scope of the thesis.

Operational Phase. The design process ends as the operational phase begins, but part of a completed design is documentation describing system operation and maintenance. In addition, the final design should include features helpful to the system users: easy, reliable operation; easy, inexpensive repair; and easy modification.

Iterations Between Phases. The straight line hierarchy of the digital system life cycle is somewhat misleading. Iterations between consecutive phases is normal and was done regularly in the time digitization system design. For the most part, changes to documents from a completed phase were limited to the immediate predecessor of the current phase. This simplified the design effort, and shows the effectiveness of the system life cycle design approach.

Overview of the Thesis

This paper presents complete documentation for requirements definition, system design, hardware selection, and software structure. Circuit layout for all but a few specialized functions is provided. However, because system specifications may change as a result of this

investigation, only critical segments of code are developed. Testing and integration are discussed briefly.

Following the life cycle sequence outline, the requirements definition and the system design are presented in Chapters II and III. To maintain logical continuity, however, hardware selection and circuit layout are combined as are software structure and code. Chapter IV gives the hardware selection and circuit layout for the digital processor and its support chips, and Chapter V does the same for functions which must be implemented in special purpose hardware. The software structure and some sample flow charts outlining coding requirements are presented in Chapter VI. The thesis concludes with results and recommendations in Chapter VII.

II Requirements Definition

In the requirements definition phase, general ideas about a new system are made specific and checked for completeness. For a digital system, performance and I/O must be defined (Ref 6:66) where performance is accomplishing specified functions and meeting timing requirements. The word definition implies the use of some language, but English prose is normally not sufficient to document design requirements because ambiguities are difficult to avoid. Thus, a more descriptive and precise language is necessary, a language which is likely to combine verbal explanations with some well defined form of diagrammatic notation. Whatever language is selected for the requirements definition must be easily understood both by the people who originated the system concept and the designers who develop the concept into a working device. The eventual system users should be able to agree that the requirements definition document describes what they really want, and the designers must be able to understand exactly what is needed. Finally, the chosen language must lend itself to easy verification of completeness and consistency. Completeness minimizes the threat that an unexpected problem will force major changes to a system late in the design process, while consistency assures that the new system is actually possible.

The objective of this Chapter is to develop a requirements definition which is specific, understandable, and complete. To accomplish this objective, methods from SADT are used to build a requirements definition model. The next section explains why SADT is used. The following section completes the chapter by presenting the SA activity model.

Requirements Definition Language

Structured Analysis (SA) has most of the prerequisites for a good requirements definition language. It uses precise, well defined graphical notations which have been refined over several years of use. The diagrams are simple and easily understood by people with a scientific background, and because SA models are built top-down they tend to be complete. Other well developed requirements definition languages exist (Ref 7:7), but they are computer based rather than manual as SA is, and they would be too expensive and complex for a project as small as the time digitization system. One notable deficiency in the SA diagrams is the lack of a method to specify timing relationships between several functions, but the defect is not critical in this application.

Structured Analysis conventions are described in several publications produced by SofTech (Refs 8;9), and Appendix A gives a short review of the major conventions. However, in this paper a few departures from normal SA procedures are made. Most notable is the omission of a data model corresponding to the activity model. During the time digitization system design process a data model was prepared,

but it provided only a few minor insights. The data model is not included in the paper to simplify the presentation. Text describing each diagram is incorporated into the chapter, and the diagram orientation is changed to fit the style of the thesis. Timing specifications and interface (I/O) requirements are added to the text for each diagram to make the SA model a more complete requirements definition.

The Structured Analysis activity model presented in this chapter is the fourth version of the model. After the first version was prepared, it was reviewed for clarity and consistency by a reader familiar with SA conventions but not with CIGTF needs. The model was then revised and sent to the Guidance Division to be evaluated for functional accuracy. The evaluation identified at least one significant functional misconception which was corrected in the third version. The third model was reviewed and approved by the Guidance Division, however, a few small changes and corrections were made later in the design process leading to the fourth and final model which is given here.

Structured Analysis Activity Model

As an introduction to the Structured Analysis activity model, the distinctive features of the time digitization system are reviewed here. The function of the time digitization system is to acquire data during tests of inertial components, to format the data, and to transmit the data directly to a minicomputer. Test data from the inertial hardware is provided by several sets of signals, with each set containing two

channels of digital pulse trains and up to eight channels of analog signals. From this point on in the design, a time digitization system is assumed to process only one pair of digital pulse trains and their associated analog channels. Each additional set of signals can be processed by an identical time digitization system, and the only necessary relationship between time digitization systems is a common elapsed time clock.

The data which a time digitization system must collect depends on the pulses on the two digital channels. One of the channels is designated the primary or master channel, and the other is used as a reference channel. A continuous count, called a net count, which relates directly to changes in angular velocity in the inertial hardware is computed using pulses on the primary channel and their phase relationship to pulses on the reference channel. Then, at certain pulses called events on the primary channel, the net count and the value of the analog signals must be stored as data samples. Several modes for selecting events are described in the SA model; although the purpose for each mode is beyond the scope of this paper, each method for collecting data is appropriate for different test conditions.

The data samples collected by the time digitization system must be organized in a standard format and collected into groups of 32 which are called data records. The time digitization system is to transmit data to the minicomputer a record at a time at a rate of 122 records/second (approximately 1 million bits/second). The 122 record/second

Node	Title
A-1	Perform Guidance Equipment Testing
A-0	Acquire Test Data
A0	Acquire Test Data
A1	Condition Digital Signals
A2	Determine Net Count
A3	Identify Selected Event
A32	Skip N Events
A33	Delay for Time Window
A4	Prepare Data Record
A44	Prepare Data Sample
A45	Send Data Record

Figure 2-1 Requirements Definition Model Index

rate is also the maximum instantaneous or peak data acquisition rate although the maximum continuous acquisition rate is 8 records/second.

The last few paragraphs describe the time digitization system in general. The purpose of the SA activity model is to define in detail the requirements imposed on the time digitization system. An index to the model is provided in Figure 2-1 and can be used as an overview to the functions the system must perform.

The text describing Node A-1 begins on page 18. From that point on in this chapter, the text for each node is on a separate page which faces the figure showing the node.

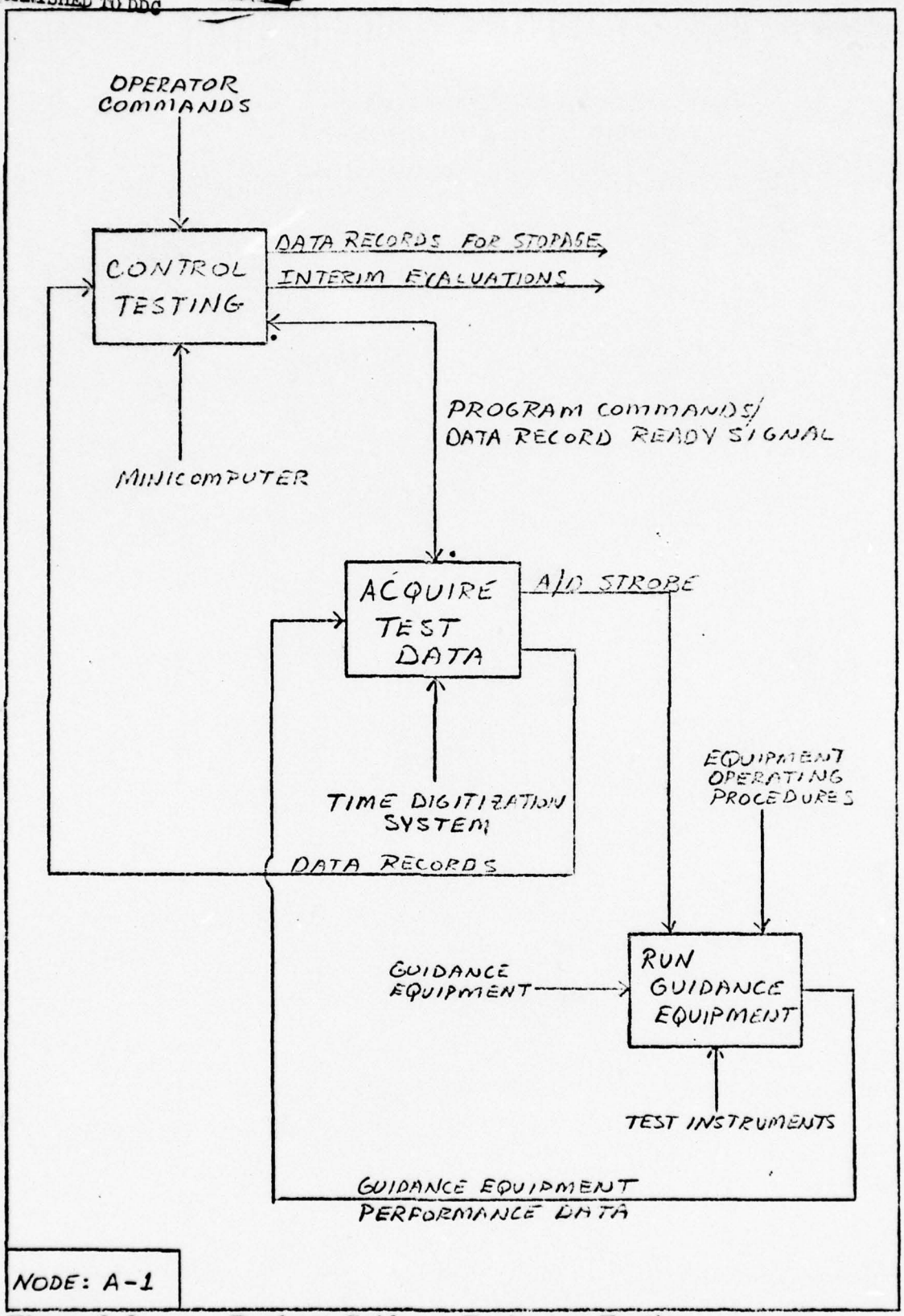


Figure 2-2 Perform Guidance Equipment Testing

Perform Guidance Equipment Testing (Node A-1). Figure 2-2, Perform Guidance Equipment Testing, presents the context in which the time digitization system must operate. The immediate concerns during a test of guidance equipment are operating the guidance equipment (3), acquiring test data (2), and controlling the test process (1). The diagram shows the time digitization system (2M1) both as an interface between the test instruments (3M1) and the minicomputer (1M1), and as a processor which operates in parallel with the minicomputer.

The minicomputer is to be a model in the HP 2100 series, and communication between the minicomputer and the time digitization system must be done via a 16 bit bidirectional data bus. The data record ready signal (2C1) may be used for interrupts or direct memory access in the minicomputer. Analog-to-digital (A/D) converters are assumed to be part of the test instruments and require a strobe (201) to start signal conversion.

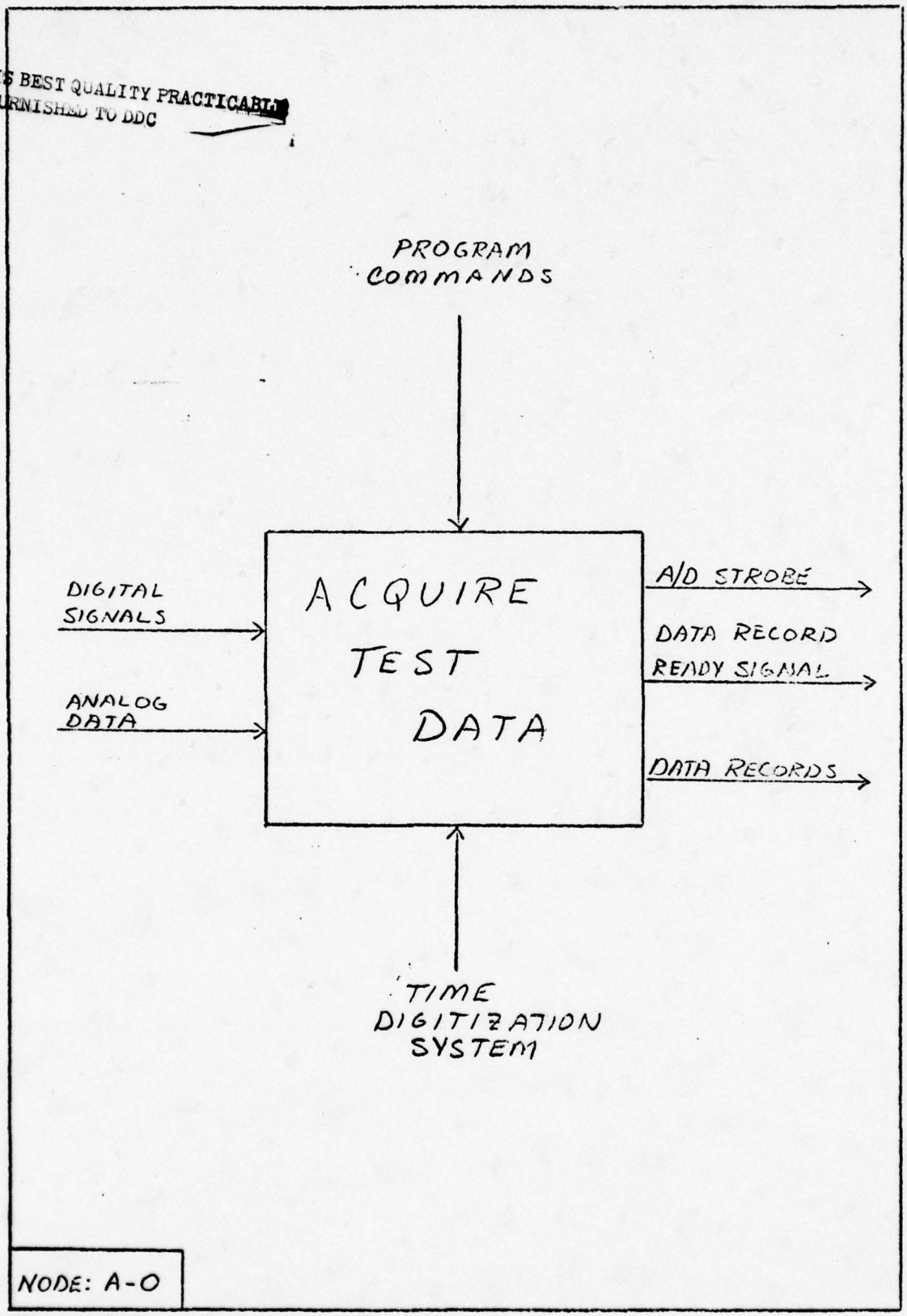


Figure 2-3 Acquire Test Data

Acquire Test Data (A-0). Node A-0, Figure 2-3, begins the model of time digitization system functions. The purpose of the model, as noted before, is to define the system requirements; the viewpoint of the model is that of the system designer. The system acquires data according to the method specified by the program commands (C1). Data input comes in the form of continuous streams of digital pulses (I1) and analog data (I2) from A/D converters. Output includes a strobe for the A/D converters (01), a data record ready signal (02), and the data records themselves (03).

The system must be able to acquire and transmit a continuous average of 8 data records per second. For bursts of input, the system must be able to reach a maximum instantaneous acquisition rate of 122 records per second. Transmitting stored records must be done at a minimum rate of 122 records per second (999,424 bits per second).

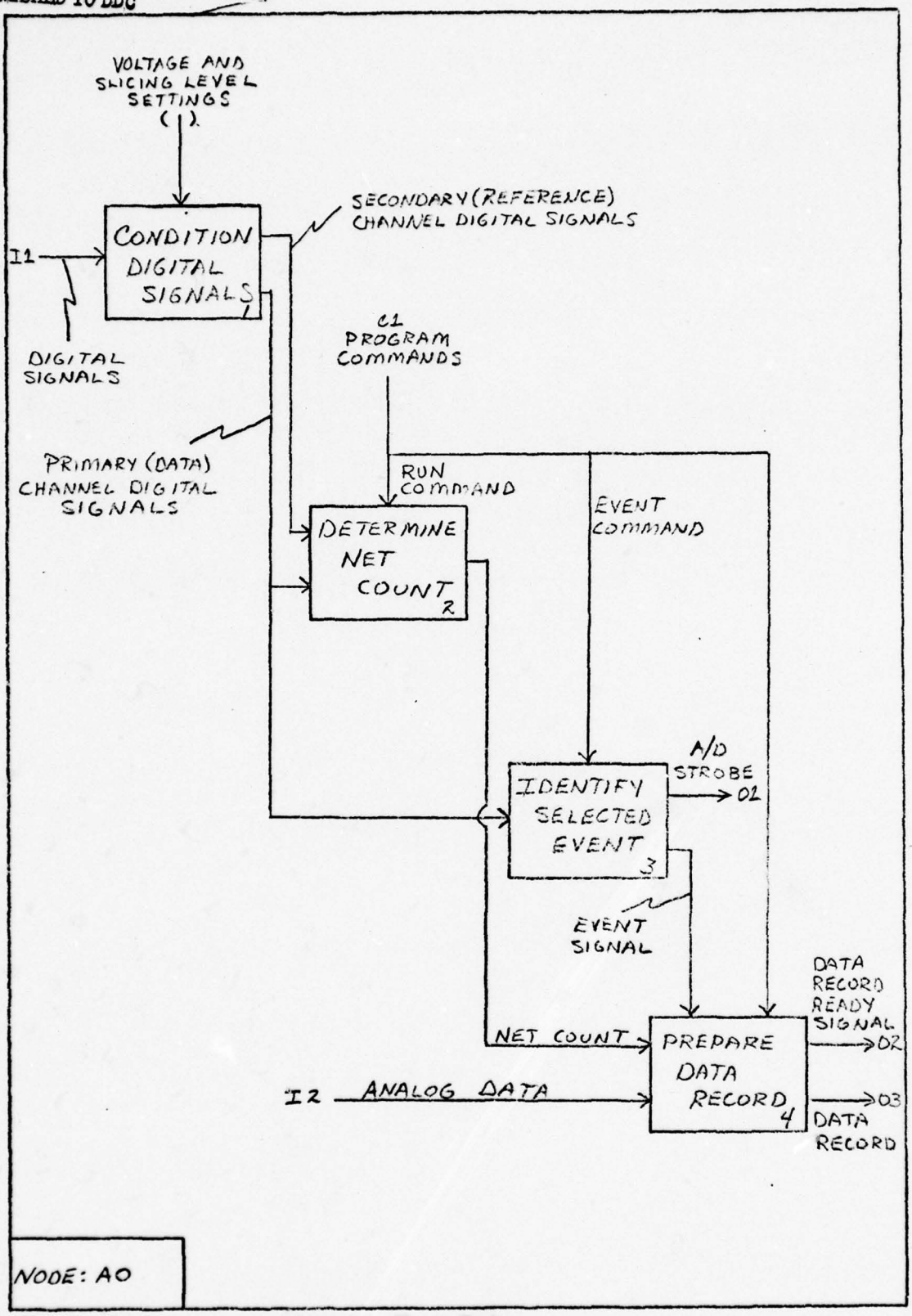


Figure 2-4 **Acquire Test Data**

Acquire Test Data (A0). Node A0, Acquire Test Data, which is shown in Figure 2-4 decomposes the operation of the time digitization system into four primary functions: the incoming digital signals (I1) must be conditioned (1), that is made compatible with the electronic devices used in the remainder of the system; a net count must be computed (2); events must be identified (3); and all data collected must be formed into records (4). Condition Digital Signals (1) provides two channels of pulses, a primary (102) carrying test performance information and a secondary (101) which is used as a reference. The two channels of pulses are used to determine the net count (2), and the primary channel is used to determine when an event of the type specified by the event command (3C1) has occurred. An event signal (302) causes data to be collected to become part of a data record. Voltage and slicing level settings (1C1) first appear at this level of the model because they are envisioned as manual controls which will be adjusted once before a test begins.

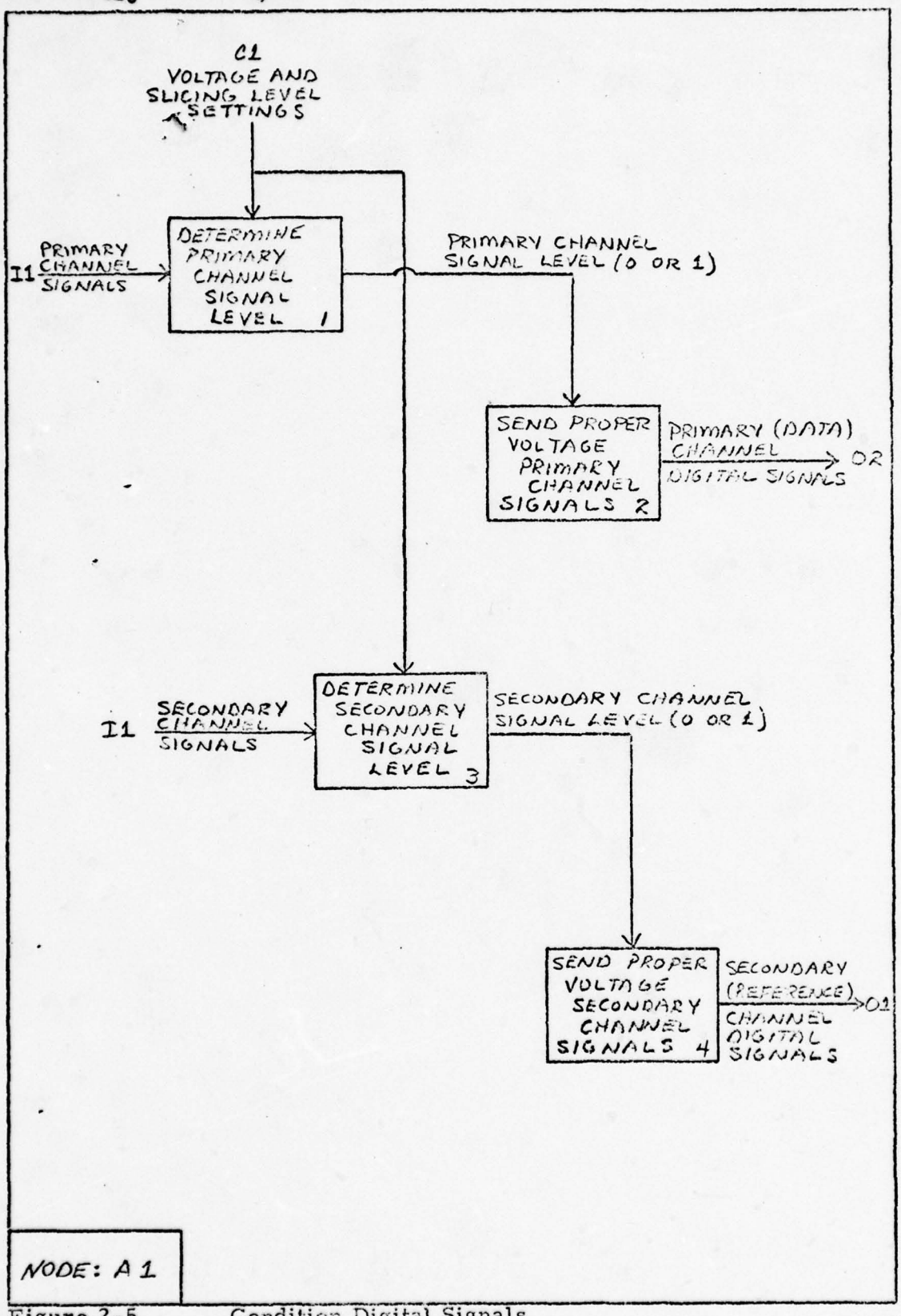


Figure 2-5 Condition Digital Signals

Condition Digital Signals (A1). Condition Digital Signals, Node A1, is presented in Figure 2-5. The function described in this diagram is converting digital signals (I1) from laboratory test equipment into signals (01, 02) which are compatible with circuitry in the rest of the system. The voltage level setting (C1) should reflect the voltage range of the incoming signals, and the slicing level (C1) determines the difference between a logic zero or one signal.

Incoming digital signals (I1) are to meet the following timing criteria:

Pulse Frequency	.01 Hz to 100 KHz per channel
Minimum Pulse Width	1 microsecond

Output digital signals (01, 02) must meet the following criteria:

Slicing Delay	0.2 microseconds maximum
Signal Rise Time	10 to 100 nanoseconds

Incoming digital signals (I1) are to have the following characteristics:

Signal Range	2 to 20 volts
DC Bias	$\pm$ 20 volts
Impedance	1 Kohm minimum

The slicing level setting must be variable between 1.5 and 4 volts.

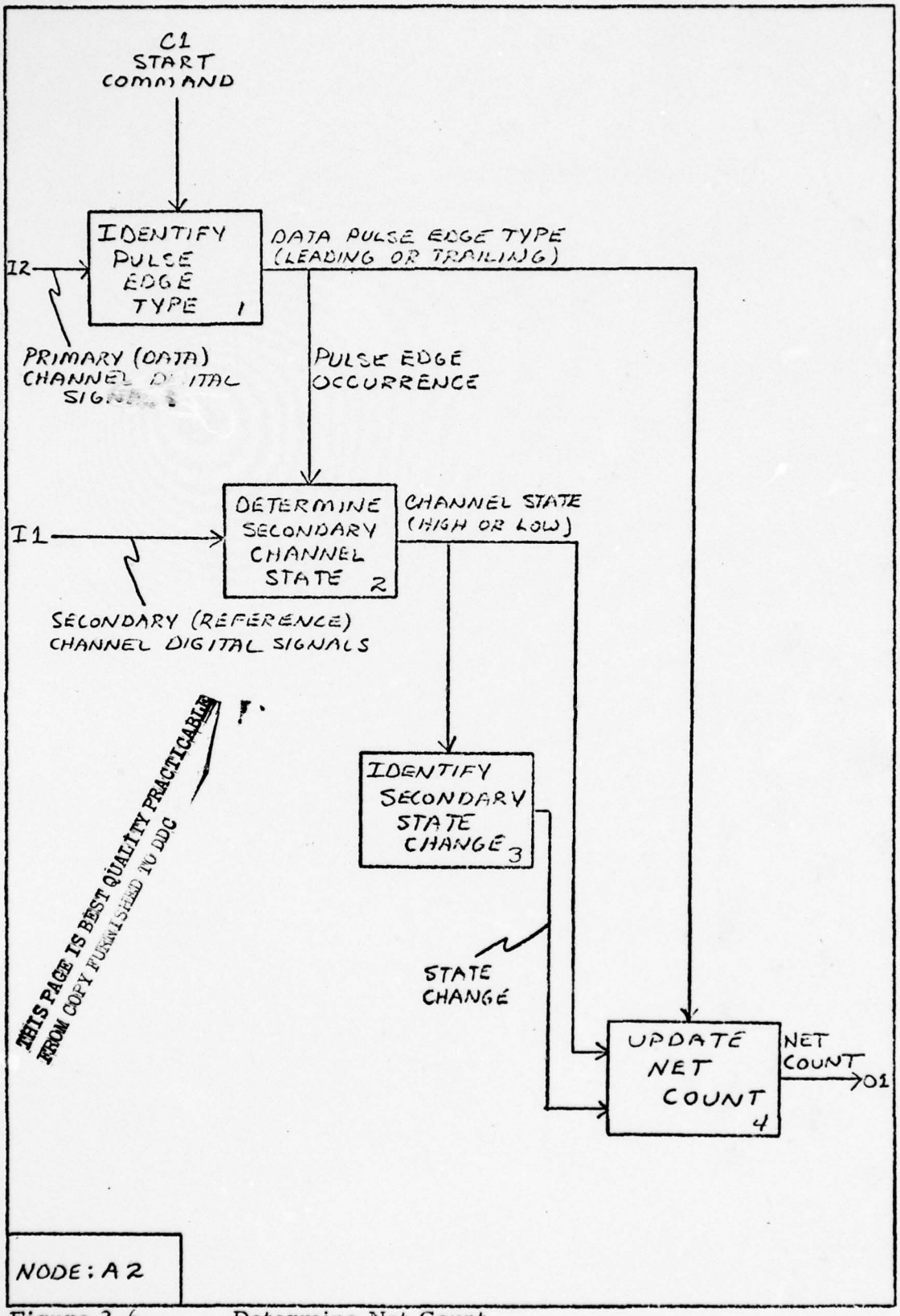


Figure 2-6 Determine Net Count

Determine Net Count (A2). The process of determining the net count is presented in Figure 2-6. The net count (01) provides a continuous record of the phase relationship between the primary channel digital signals (I2) and the secondary or reference channel digital signals (I1). The primary channel signals represent a performance parameter (angular velocity) of an inertial component being tested, while the reference channel provides a standard against which the performance can be measured. Factors which determine a change in the net count are a pulse edge (leading or trailing) on the primary channel (101), the state of the secondary channel (201), and a state change on the secondary channel (301) since the last pulse edge on the primary channel.

A flow chart demonstrating how to compute the net count is presented in Figure 2-7. In the figure, X represents the primary channel, and Y stands for the state of the reference channel. Figure 2-8 gives a sample net count computation.

Logic equations can also be used to specify changes in the net count as follows:

L = Leading Edge
T = Trailing Edge
S = Secondary Channel High
C = Secondary Channel State Change

$$\text{Increment Count} = L\bar{S}C + TSC \quad (1)$$

$$\text{Decrement Count} = LSC + T\bar{S}C \quad (2)$$

At all other times, the net count must not be changed.

Since the net count (01) may change at any pulse edge on the primary channel, and because the minimum pulse width is one microsecond, the net count must be updated in less than one microsecond. The net count is to run continuously through a test, starting at zero only at the start command (C1). The start command may be implicit in other commands.

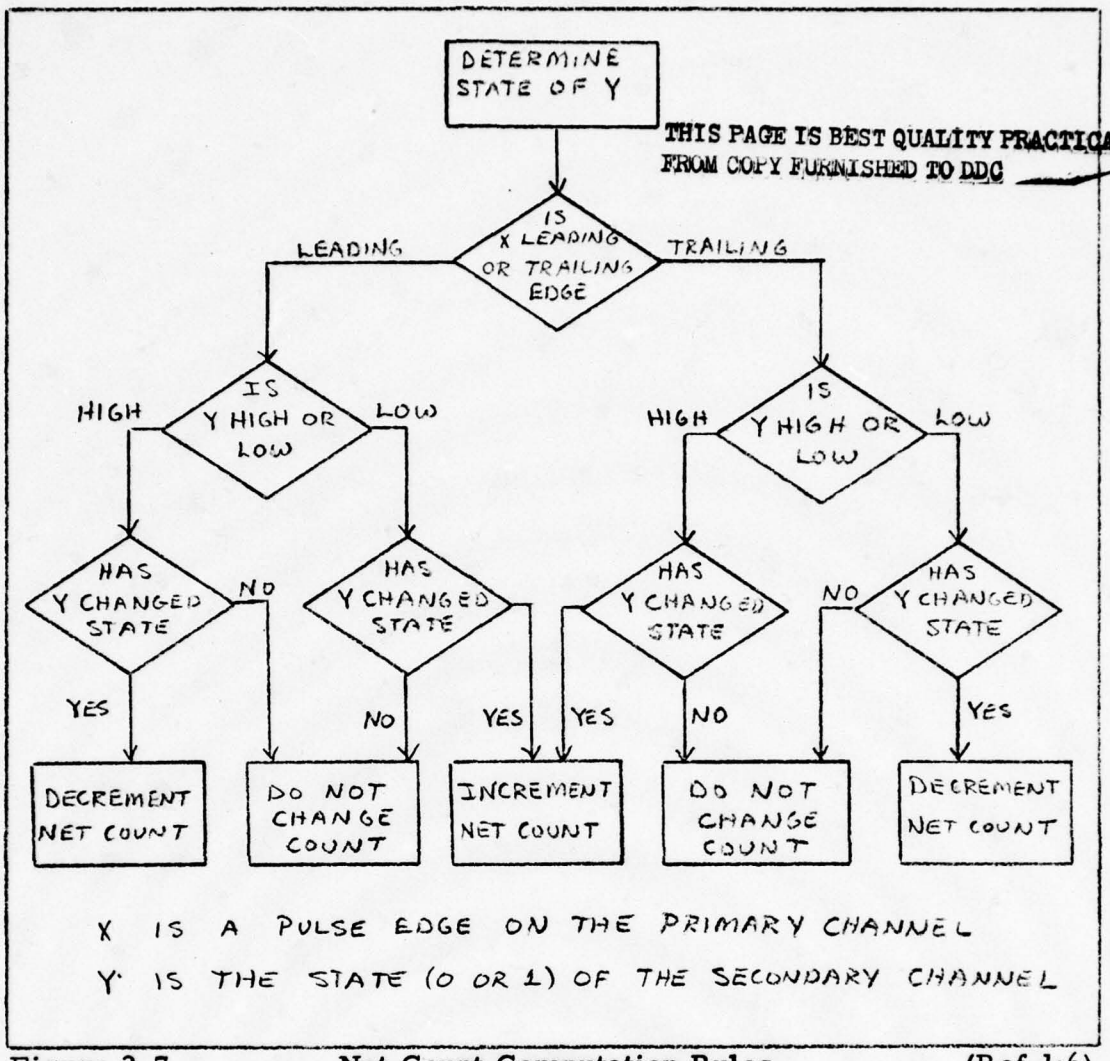


Figure 2-7 **Net Count Computation Rules** **(Ref 1:6)**

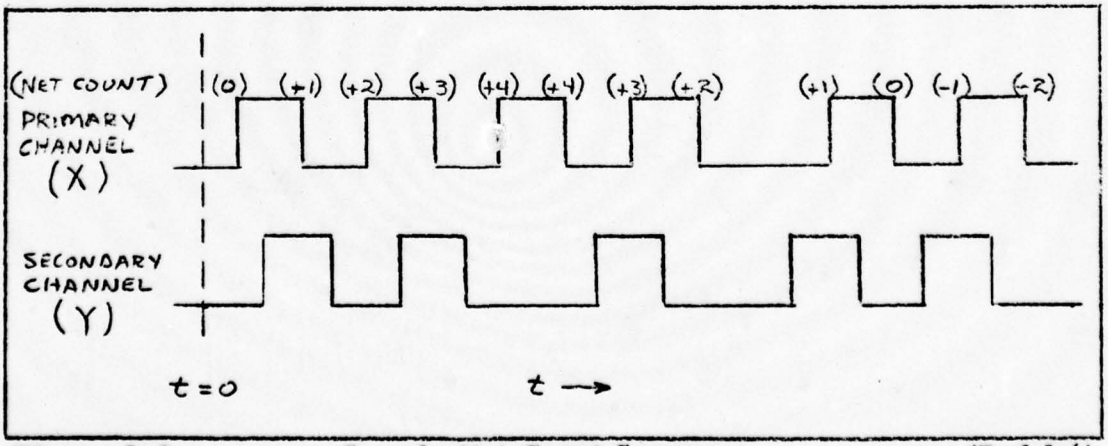


Figure 2-8 **Sample Net Count Sequence** **(Ref 1:6)**

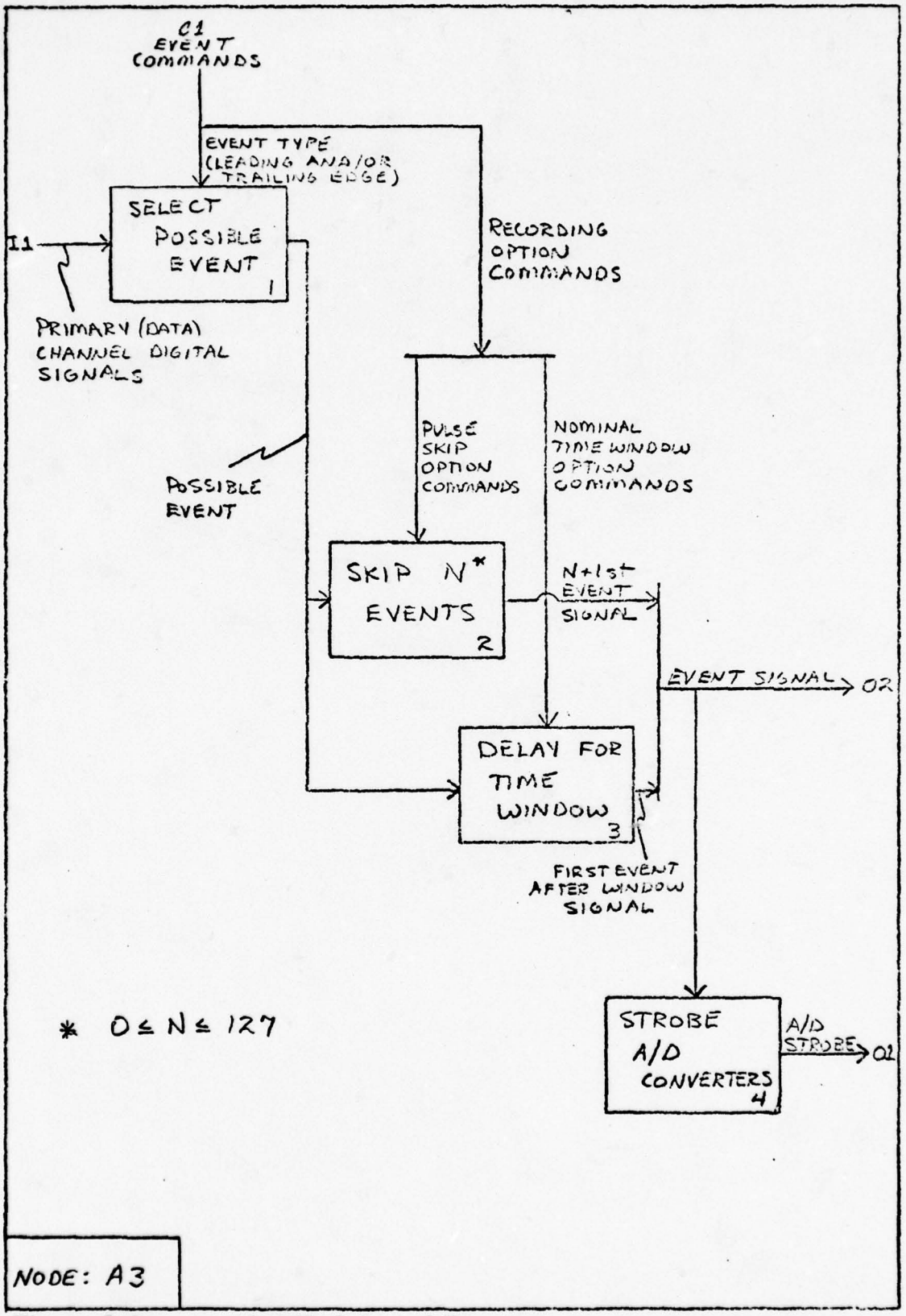


Figure 2-9 Identify Selected Events

Identify Selected Events (A3). Figure 2-9 shows the functions necessary to determine when an event has occurred. An event is a specified pulse edge on the primary channel (I1), and at the time of an event, a signal (02) must be sent to initiate storage of data by the next module. Identifying an event in the manner specified by the event commands (C1) begins with detecting a leading and/or trailing edge of a pulse (101). Once a possible event is selected, one of two options can be used to determine if an event signal should be generated. The first option is to skip N events (2), and the second is to delay for a nominal time window (3). At the N+1st possible event or at the first possible event after the end of the nominal time window, an event signal is issued which also causes the generation of a strobe to the analog-to-digital converters (4).

The selected event must be identified within the resolution of the master clock, and the clock frequency is to be 40 MHz. The A/D strobe must occur soon enough to allow A/D conversion within 5 microseconds of an event. A leading edge, a trailing edge, or both may be selected as events by an event command. A command must also specify the recording option to be used, either pulse skip or nominal time window.

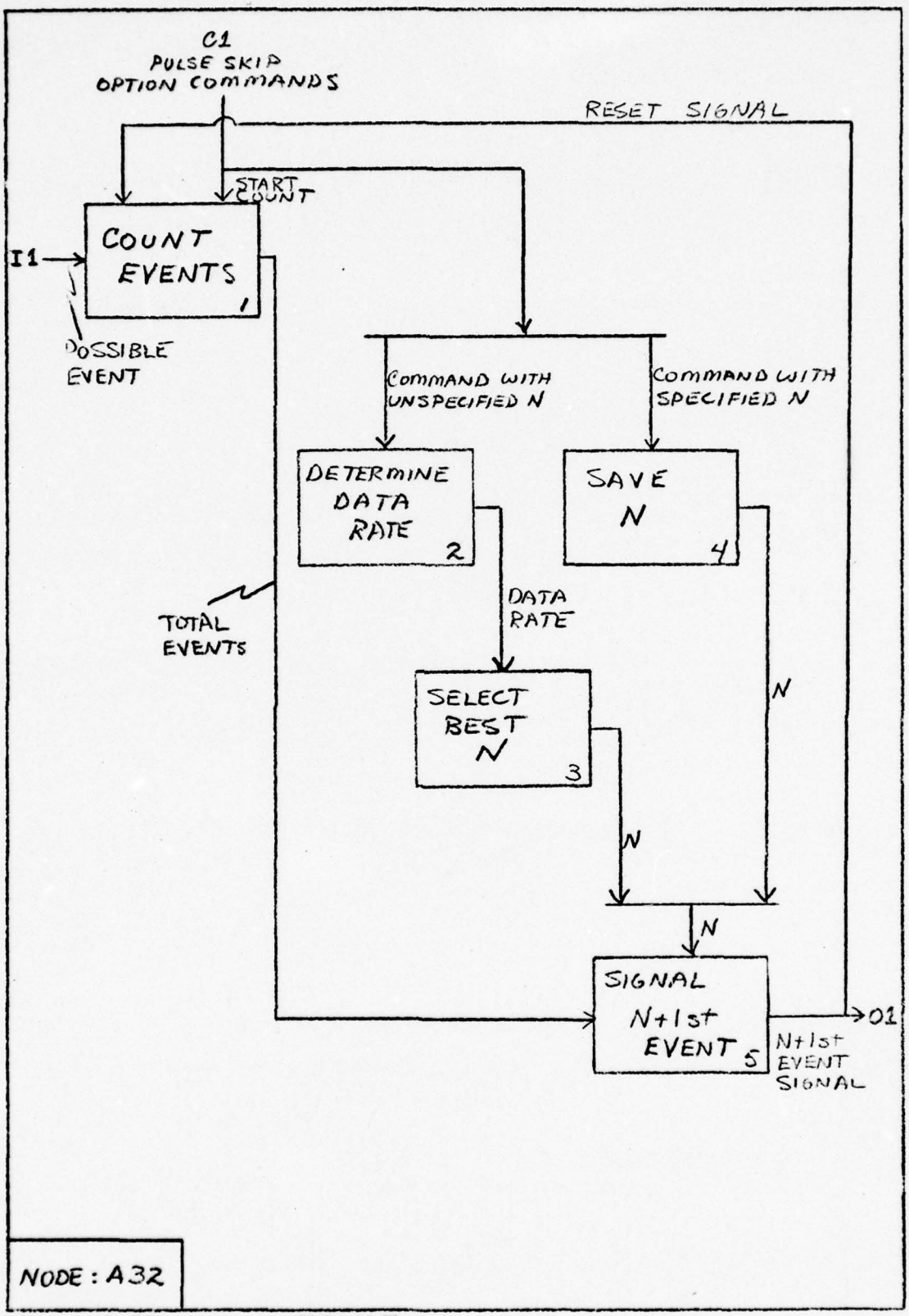


Figure 2-10 Skip N Events

Skip N Events (A32). The functions of the pulse skip option are diagramed in Figure 2-10, Skip N Events. This option is performed by counting possible events (1), comparing the total events against the N being used, and sending an event signal (5) when the total events exceed N. Counting events begins again after the N+1st event (1C1). If the pulse skip option commands do not specify an N, then the time digitization system should select an N based on the data rate so data is acquired at the maximum continuous rate.

When N is specified, its value will be such that $0 \leq N \leq 127$. When both leading and trailing edges are selected as events in node A31, N must be zero.

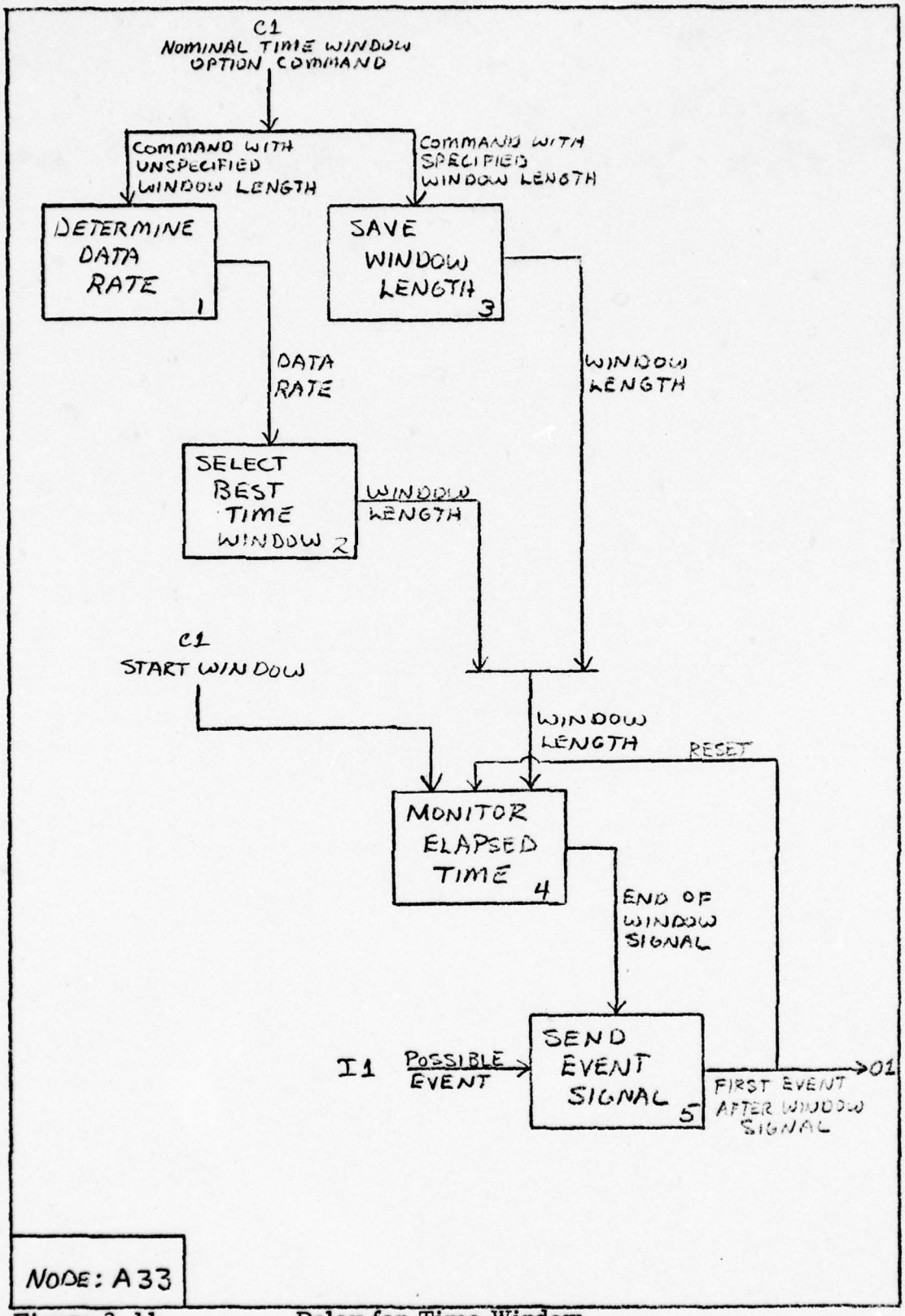


Figure 2-11 Delay for Time Window

Delay for Time Window (A33). The delay for a nominal time window, Figure 2-11, is accomplished by a timing device (4) which is preset with a window length (4C3). When the time window ends, the next event causes an event signal (01) and resets the timing device (4C1). If the nominal time window option command does not specify a window length (1C1), then a window length must be selected to keep data acquisition at the maximum continuous rate. A window length when specified will be between 10 milliseconds and 10 seconds.

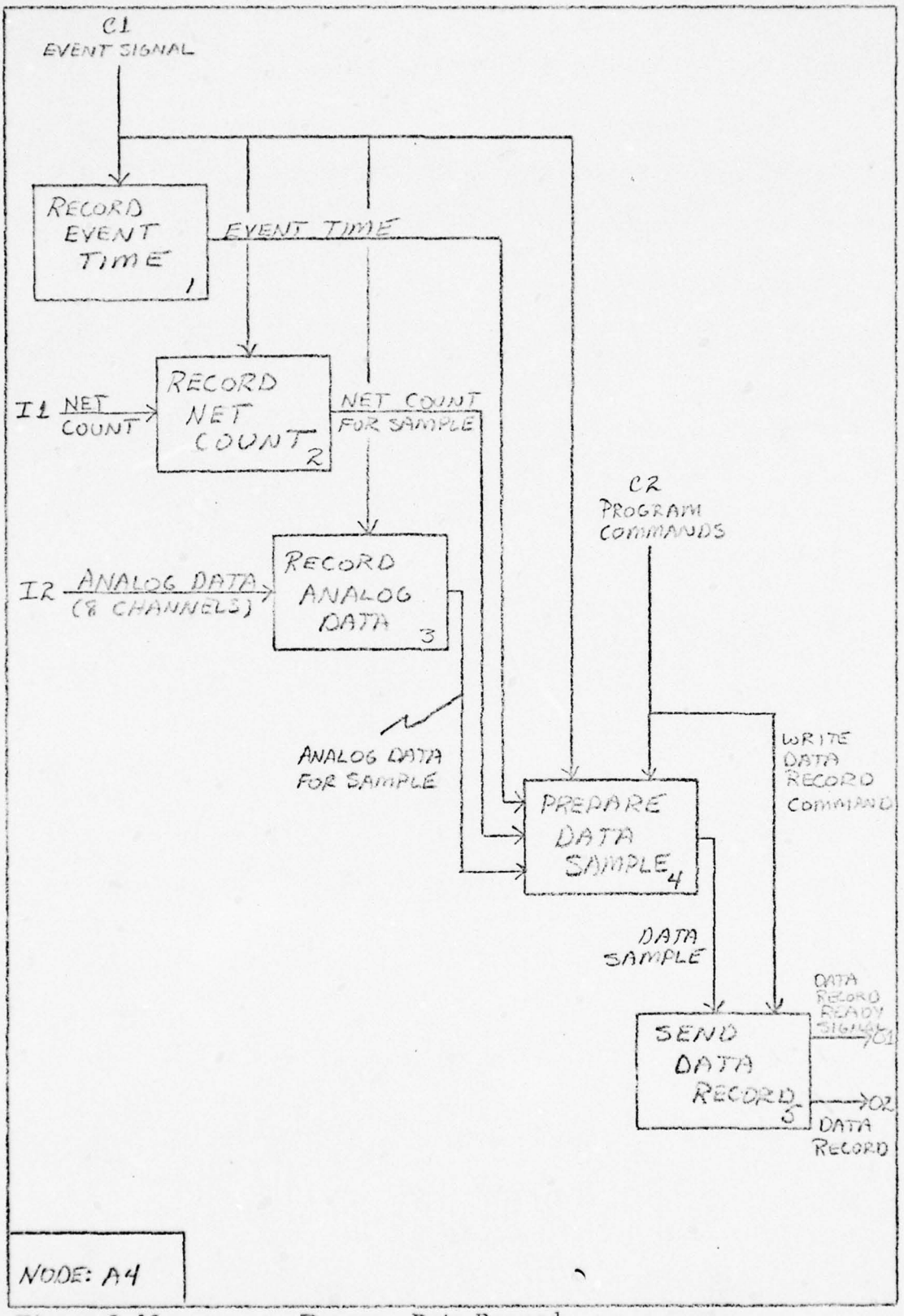


Figure 2-12 Prepare Data Record

Prepare Data Record (A4). The preparation of the time digitization system output, data records, is presented in Figure 2-12. The data sample (401) is the basic unit of information in the data record (02). Each data sample consists of the time an event occurs, which is the time digitization value (101), the net count following the event (201), and analog readings taken at the time of the event (301). Information from the program commands (C2) is used to identify the type of data in the sample. Data samples are grouped (5) and sent as data records (02).

The event time (101) is the reading from an elapsed time clock at the time an event occurs. The elapsed time clock is to be a 36 bit counter which is incremented at each pulse from the system clock. Periodically the elapsed time clock will overflow, but this is considered normal, and no provisions are necessary to indicate an overflow has occurred.

Specifications for the system clock are as follows:

Frequency	40 MHz
Static Frequency Drift	.001 ppm maximum in 8 hours
Dynamic Frequency Drift	0.5 microseconds maximum in 5 min.

Analog data (I2) is provided by up to eight analog-to-digital converters located with test instruments monitoring the inertial components under evaluation. The A/D converters are to have 12 bit resolution.

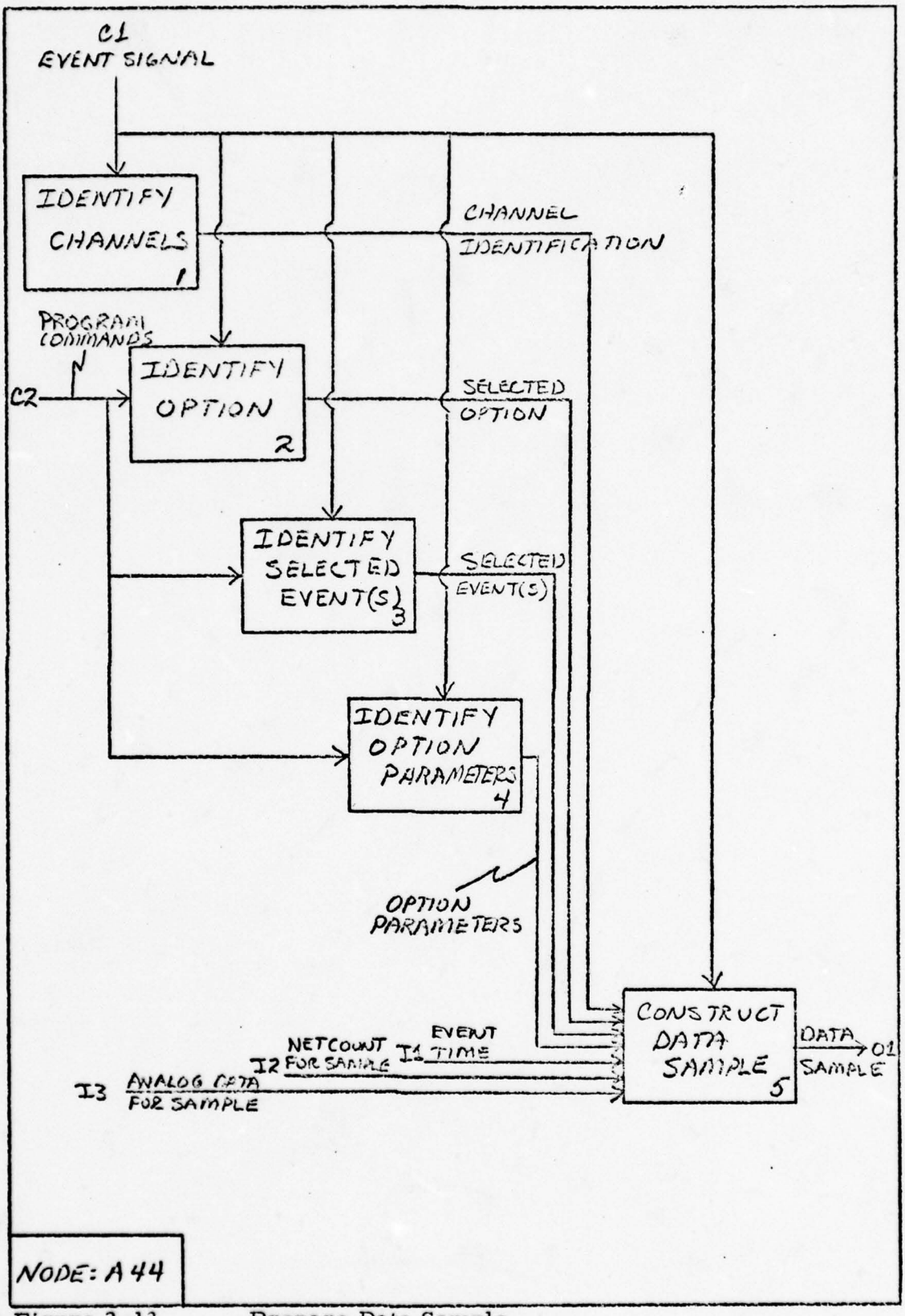


Figure 2-13 Prepare Data Sample

Prepare Data Sample (A44). Figure 2-13 shows the elements which make up a data sample. At each event signal (C1) one data sample is prepared. The channel identification (101) is a constant for each time digitization system. Program commands (C2) supply the recording option (201), selected event (301), and the option parameters (401). The event time (I1) and the net count (I2) are provided by previous modules, and the analog data (I3) comes from the A/D converters.

A data record is to be composed of 32 data samples. Thus, the processing requirement rates of node A-0 can be translated into 256 data samples continuously and 3904 data samples instantaneously.

Data samples may contain up to 16 words of 16 bits each (256 bits total). In addition, each data sample must contain the following information organized in a standard format:

Channel Identification	Primary and Secondary Channels (Fixed for each time digitization system)
Selected Option	Pulse Skip or Nominal Time Window
Selected Event	Leading Edge, Trailing Edge, or Both
Option Parameters	N or Nominal Time Window Value
Event Time	36 Bit Value
Net Count	16 Bits
Analog Channel Data	Up to 8 Readings of 12 Bits

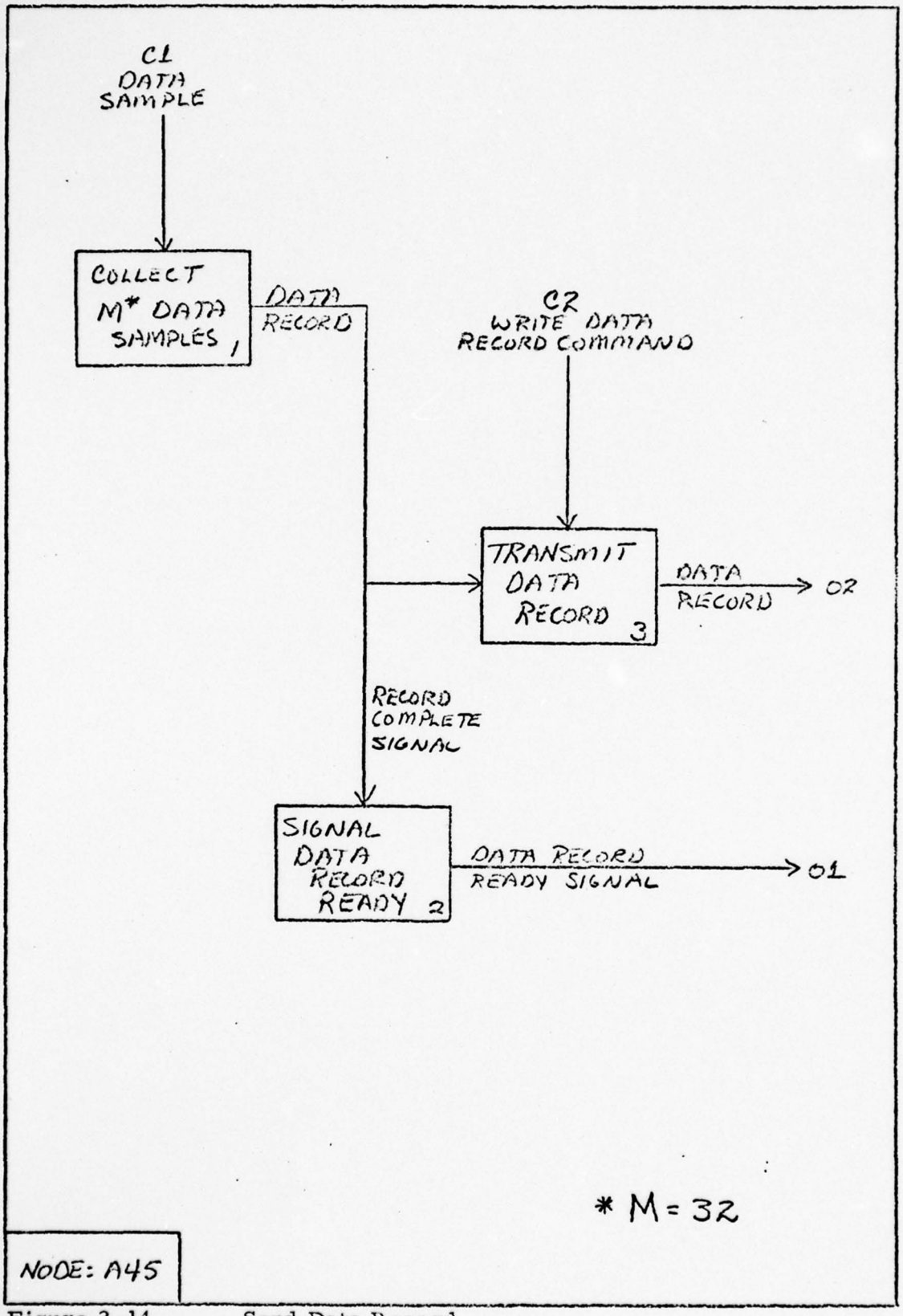


Figure 2-14 Send Data Record

Send Data Record (A45). The last function of the time digitization system, sending data records, is shown in Figure 2-14. When M = 32 data samples have been collected, a data record is formed (1). At this time the minicomputer may be notified that a record is ready (2). When the minicomputer is ready to accept a record, a write data record command (C2) is used to initiate data transmission (3). The parameter M has been used for the fixed value 32 to add flexibility in the design. As noted in node A-0, the maximum transmission rate must be 122 records per second, however, only one data record need be transmitted at a time.

The objective of this chapter was to develop a requirements definition for the time digitization system. The SA type model just completed fulfills that objective in a specific, understandable, and complete form. This model now becomes the starting point for the system design phase which is described in the next chapter.

III System Design

In the system design phase, the functions identified in the requirements definition are allocated hardware and software implementations. Once the requirements definition model had been constructed, it was obvious that some type of LSI processor was necessary because of the large amount of data to be manipulated. Several processor implementations seemed possible; among them were a single board computer, a bit slice microprocessor, a microprocessor with special purpose hardware, or two microprocessors in a master-slave configuration. The problem was to determine which processor implementations would work, and which one was most efficient.

To solve the problem, it was necessary to find a method to convert the requirements definition model into another model. In the second model, the activities would be grouped in ways that correspond to functions which can be performed by LSI or other hardware devices. The purpose of this chapter is to explain how the system design method was selected, to show how it was applied, and finally to construct the system design model.

System Design Method

A number of methods for building a system design model were tried. It was thought at one time that analyzing the data flow through the time digitization system would help identify the best type of

processor, but as mentioned before, the SA data model was of little help. As a variation on the data flow idea, a Structured Design bubble chart was developed, and then an attempt to convert the bubble chart to a program structure chart was made (Ref 10:254-300). The conversion was unsuccessful because the structure charts could not manage the asynchronous activities necessary during data acquisition and transmittal. Furthermore, several of the activities needed to be done concurrently, and this idea proved to be the key for devising a workable system design technique.

Structured Analysis models show concurrent activities; this combined with the fact that lower level nodes in the model can be regrouped easily led to the selection of SA techniques for developing the system design. By rearranging nodes into clusters of functions which must be done concurrently, the design problem can be divided into smaller parts which are independent from each other in their internal activities. The smaller design problems are more likely to fit a known hardware or software implementation, which makes the remaining design process steps much easier. Before the system design model is presented, the meaning of the word concurrency in this discussion must be explained more precisely and then applied to the requirements definition model.

Concurrent Activities

Two functions are considered to be concurrent if they must be

implemented with separate hardware devices. The concurrency can occur in two forms: the first form is for functions which must be implemented in special purpose hardware; and the second form involves software which must be executed in parallel because of timing restrictions or processor limitiations. Each type of concurrency as it applies to the time digitization system is considered in turn.

Special Purpose Hardware. A list of nodes in the SA requirements model which have timing or other specifications that require hardware implementation is given in Table I. Justification for using special purpose hardware for these functions is provided in the following paragraphs.

Table I
Functions Requiring
Special Purpose Hardware

Node	Title	Figure
A1	Condition Digital Signals	2-5
A2	Determine Net Count	2-6
A31	Select Possible Event	2-9
A34	Strobe A/D Converter	2-9
A321	Count Events	2-10
A325	Signal N+1st Event	2-10
A334	Monitor Elapsed Time	2-11
A335	Send Event Signal	2-11
A41	Record Event Time	2-12
A42	Record Net Count	2-12
A43	Record Analog Data	2-12

Node A1, Condition Digital Signals, concerns modifying signal characteristics. It is by nature a special purpose hardware task and so must be done concurrently with the other functions in the system.

Because the minimum input pulse width is given as one microsecond and because the net count may change on both the leading and trailing edge of a pulse on the primary channel, the net count must be updated in less than one microsecond. The flow chart in Figure 2-7 shows that several instructions would be required to update the net count through software, and it is unlikely that any microprocessor, even bit slice, could meet the one microsecond restriction. Hence, Node A2, Determine Net Count, also requires special purpose hardware.

Events must be identified within the resolution of a 40 MHz clock which means that several nodes subordinate to Identify Selected Events (A3) and Prepare Data Record (A4) must be performed within 25 nanoseconds and therefore in parallel with other system nodes. Select Possible Event (A31), Signal N+1st Event (A325), Monitor Elapsed Time (A334), Send Event Signal (A335), and Record Event Time (A41) are in this category.

Events may occur on the leading and/or trailing edge of pulses on the primary channel, and the net count may change on any of the pulse edges. If the leading edge is selected as an event, and the pulse width is the minimum size, one microsecond, then the net count must be computed and saved in less than one microsecond following an event. Otherwise, the net count might be changed by the trailing edge of the same pulse which constituted the event. This dictates the use of special purpose hardware for Record Net Count (A42).

The timing restrictions applied to Strobe A/D Converters (A34),

Count Events (A321), and Record Analog Data (A43) are less severe, but special purpose hardware is still the best implementation. The hardware for the A/D strobe should be almost trivial, and since the A/D conversion must be completed within five microseconds of an event, any type of microprocessor interrupt system would be too slow. The same type of reasoning applies to Count Events where the maximum time for updating the event count is ten microseconds.* Maintaining the event count in software would require almost full time use of a microprocessor, and the cost would be greater than for special purpose hardware. Recording analog data could be done by a microprocessor, but an interface which holds the data until the microprocessor is ready simplifies the design.

Processor Requirements. The remaining functions in the requirements definition model may be done through software. These nodes are listed in Table II. The first four nodes have to do with adjusting the option parameters to maintain the maximum continuous data rate, and the last two concern processing data samples and data records.

Determine Data Rate (A322 and A331) can be accomplished by computing the elapsed time between two samples and comparing that value to a standard which gives the best data rate. The difference between the actual elapsed time and the standard can then be used to

* An event count is necessary only if N> 0, and if N> 0 the selected event can be a leading edge or a trailing edge but not both. The maximum pulse frequency is 100 KHz which means that two consecutive leading or trailing edges can be no closer than 10 microseconds apart.

Table II
Software Functions

Node	Title	Figure
A322	Determine Data Rate	2-10
A323	Select Best N	2-10
A331	Determine Data Rate	2-11
A332	Select Best Time Window	2-11
A44	Prepare Data Sample	2-13
A45	Send Data Record	2-14

adjust N (A323) or the nominal time window (A332). These four functions are not time critical and can be easily performed by a digital processor.

Preparing a data sample (A44) and sending a data record (A45) involve manipulating substantial numbers of data words. Again, a digital processor is the appropriate implementation.

To determine if more than one processor is necessary to meet data processing times, the most time critical processing is considered. The maximum data processing rate of 3904 samples per second is required for acquisition alone or for transmission alone. Other functions need be performed only at the maximum continuous rate. Since some information in each data sample remains constant, less words must be manipulated during acquisition than during transmission. A data sample contains 16 words of 16 bits so each word must be transmitted in 16 microseconds to meet the 3904 sample/second rate. This is a reasonable length of time for bit slice and some of the faster conventional microprocessors, and for this reason the system design is based on only one digital processor.

Node	Title
A-1/Design	Operate Time Digitization System
A-0/Processor (Subsystem 1)	Run Data Acquisition Programs
A0/Processor	Run Data Acquisition Programs
A1/Processor	Process Program Commands
A12/Processor	Send Hardware Control Signals
A2/Processor	Form Data Sample
A3/Processor	Select Best Option Parameters
A4/Processor	Send Data Record
A-0/Hardware (Subsystem 2)	Collect Data
A0/Hardware	Collect Data
A1/Hardware	Condition Digital Signals
A2/Hardware	Control Hardware Operation
A21/Hardware	Store Hardware Control Signals
A3/Hardware	Determine Net Count
A4/Hardware	Signal Selected Event
A5/Hardware	Save Data for Sample

Figure 3-1 Design Model Index

System Design Model

Figure 3-1 gives an index for the system design model. To maintain continuity with the requirements definition model, nodes which have not been changed retain their original names, while new activities or groups of activities have been given new titles. The text describing each node does not repeat material presented in Chapter II except when necessary for clarity.

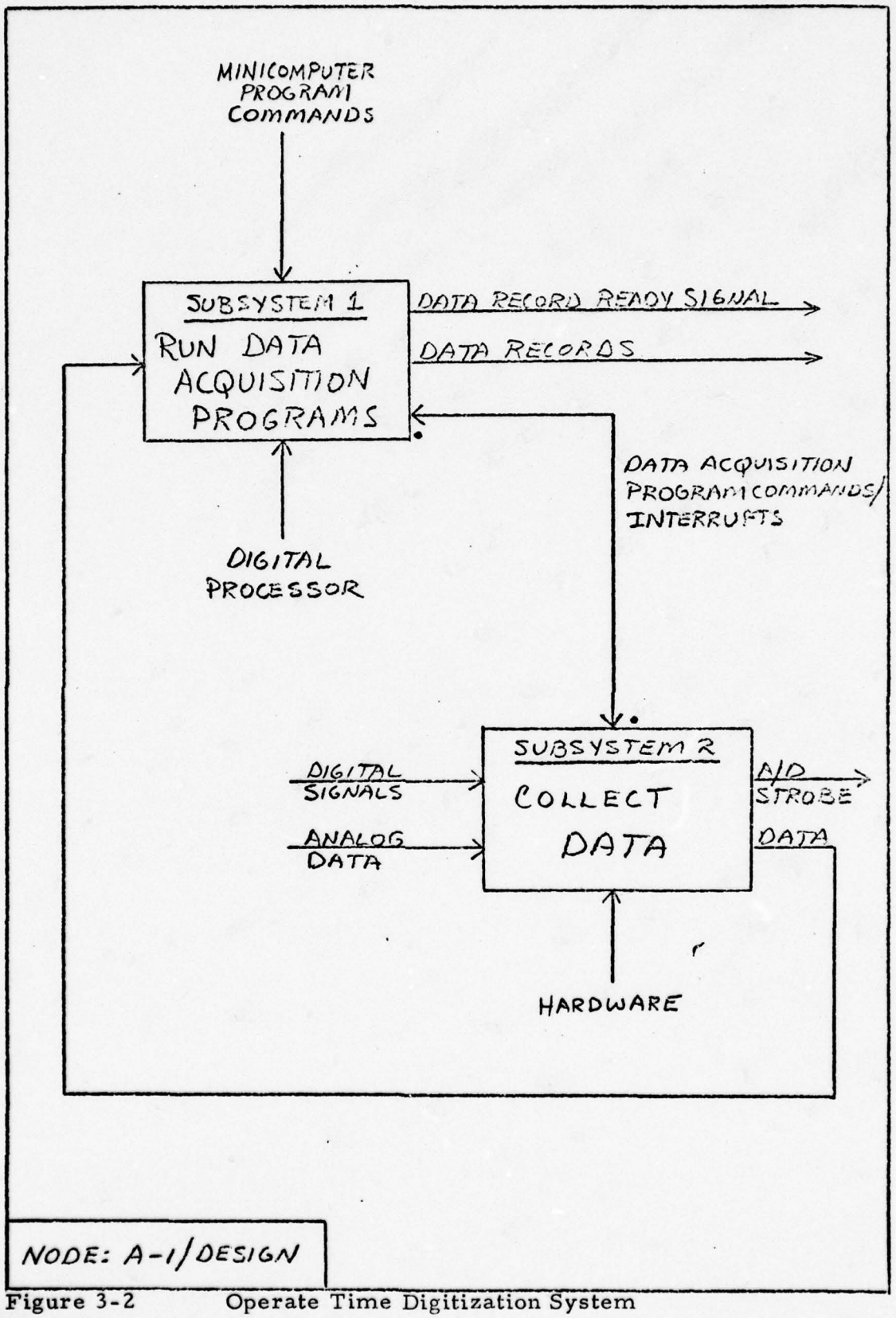


Figure 3-2 Operate Time Digitization System

Operate Time Digitization System (A-1/Design). The separation of time digitization system functions into two subsystems is shown in Figure 3-2. Subsystem 1 includes the activities to be performed by the digital processor, and Subsystem 2 contains all the special purpose hardware. The mutual control arrow between the two subsystems shows that the digital processor controls the mode of operation of the hardware, while hardware generated interrupts signal when data must be saved. In the following sections, each subsystem is developed as a separate SA model.

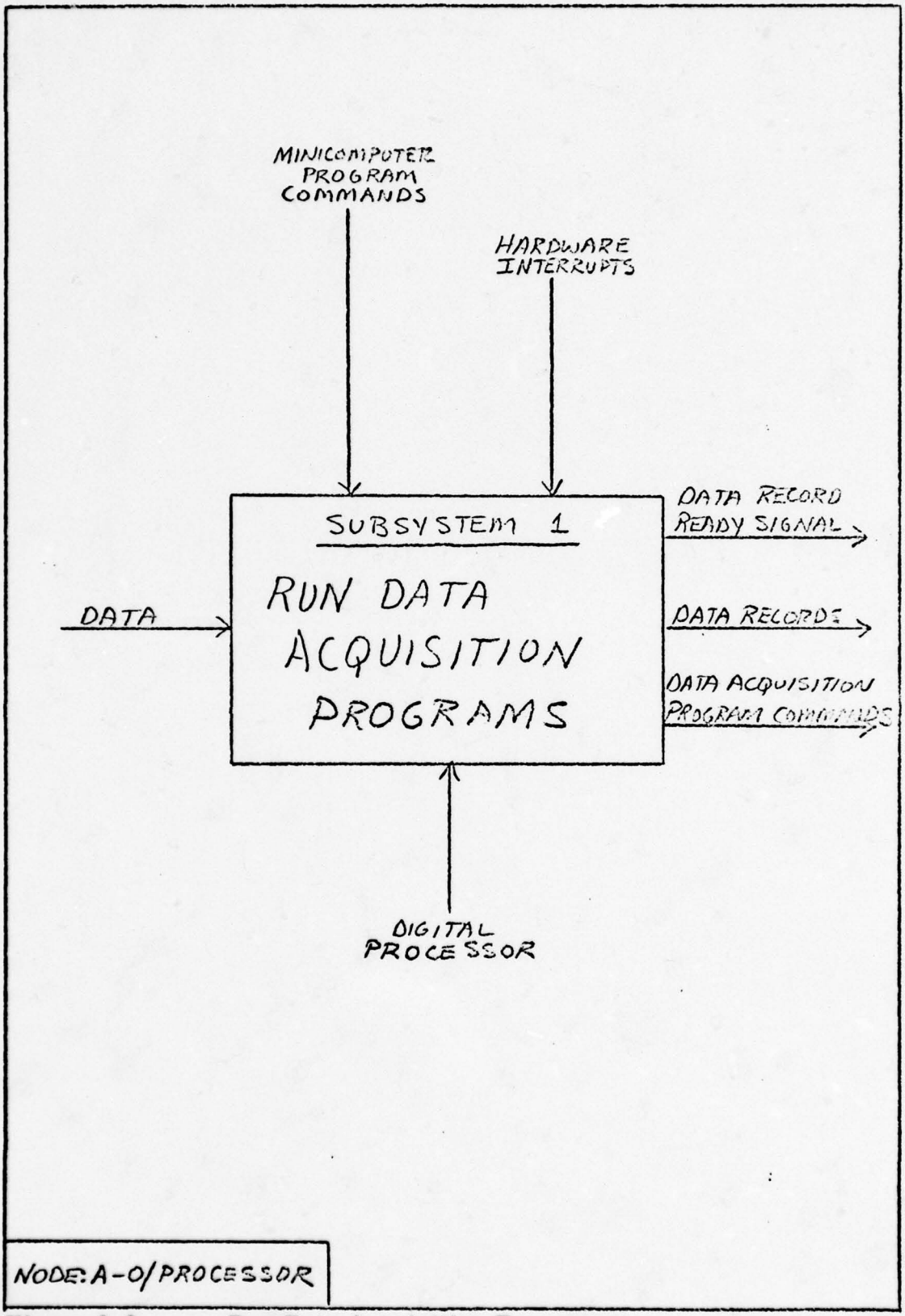


Figure 3-3 Run Data Acquisition Programs

Subsystem 1: Run Data Acquisition Programs (A-0/Processor).

The model for Subsystem 1 which defines the activities of the digital processor begins with Figure 3-3. The purpose of this processor model is to define the functions required of a digital processor; the viewpoint is again that of the system designer. In Figure 3-3 the emphasis is on the interfaces between the digital processor and a minicomputer or the special purpose hardware. All communication between the minicomputer and the time digitization system is done through the digital processor. Communication includes program commands (C1), a data record ready signal (01), and data records (02). Information passed between the processor and the special purpose hardware consists of data acquisition commands (03), interrupts (C2), and data (I1).

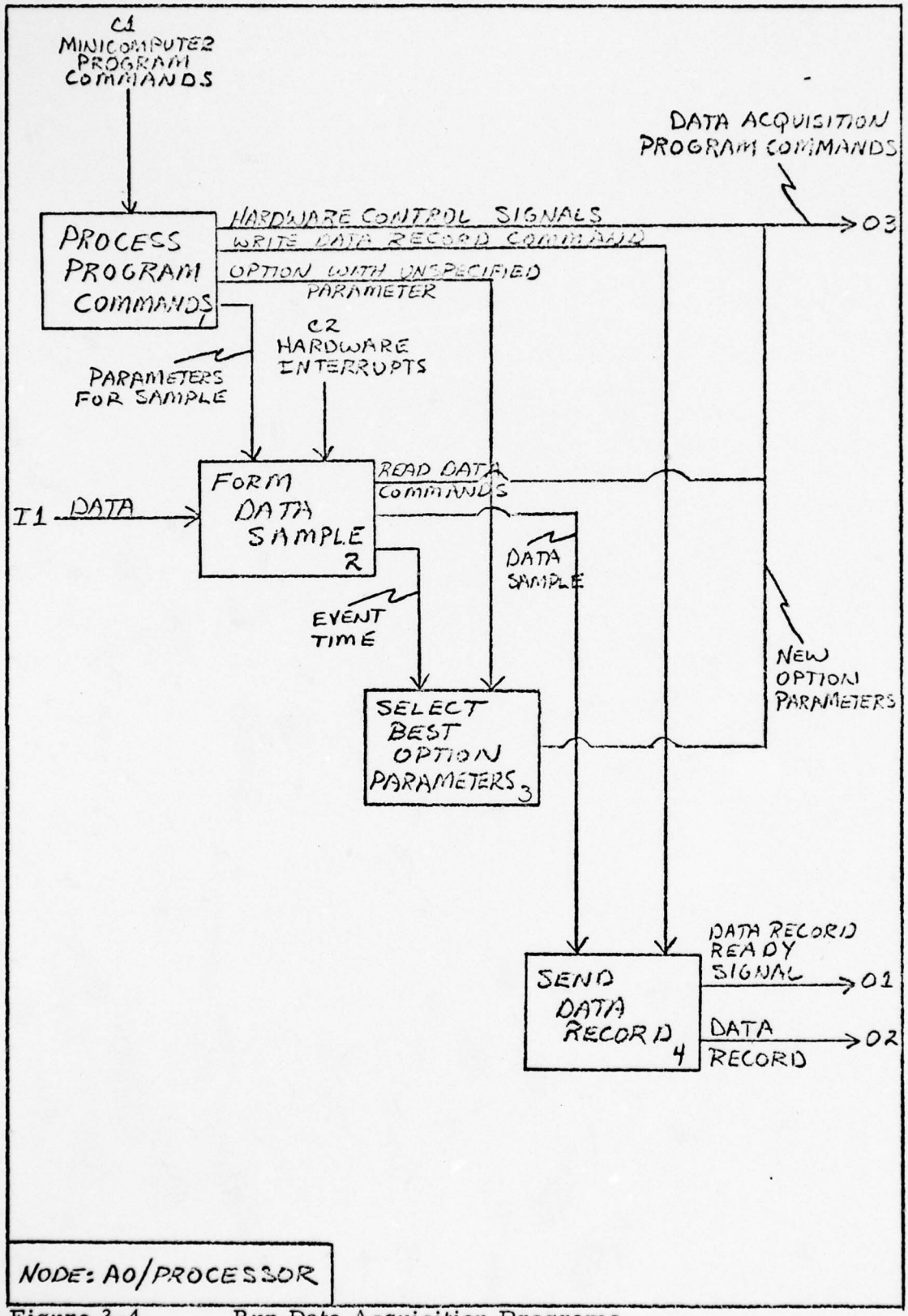


Figure 3-4 Run Data Acquisition Programs

Run Data Acquisition Programs (A0/Processor). Figure 3-4 shows that the processor must perform four major functions: process program commands from the minicomputer (1), an activity that was implicit in the requirements definition model; form data samples (2); select the best option parameters (3) when the parameters are not specified; and send data records (4). Data acquisition program commands (03) consist of control signals (101) which establish the mode of operation for the special purpose hardware and commands (201) which cause data saved in the hardware to be read into the processor.

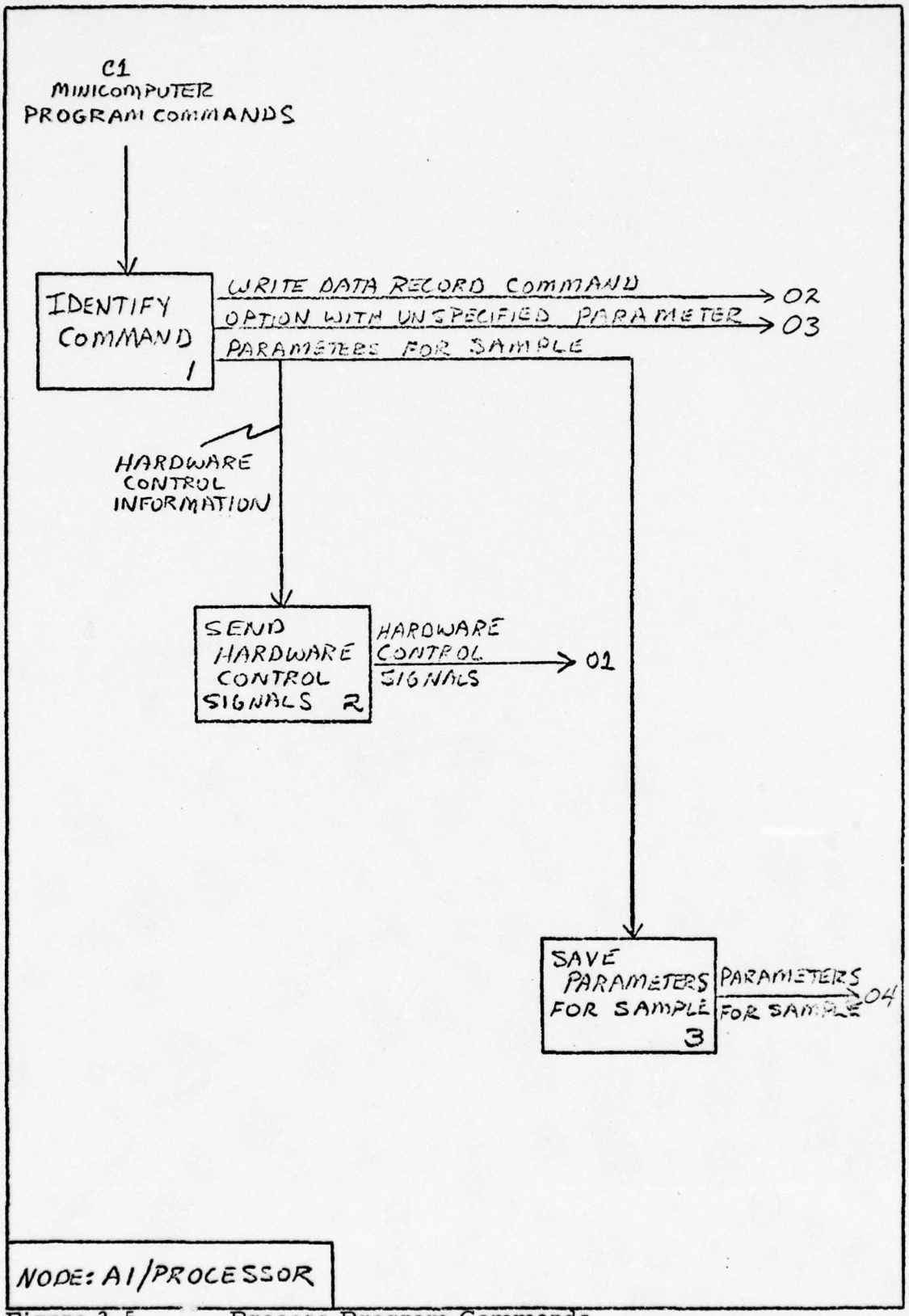


Figure 3-5 Process Program Commands

Process Program Commands (A1/Processor). The function of processing program commands is diagrammed in Figure 3-5. Here commands from the minicomputer (C1) are analyzed (1), and then other functions are activated as appropriate. Parameters which define the method to be used in collecting a data sample (103) are saved (3) and also converted into signals which control the hardware operation (2).

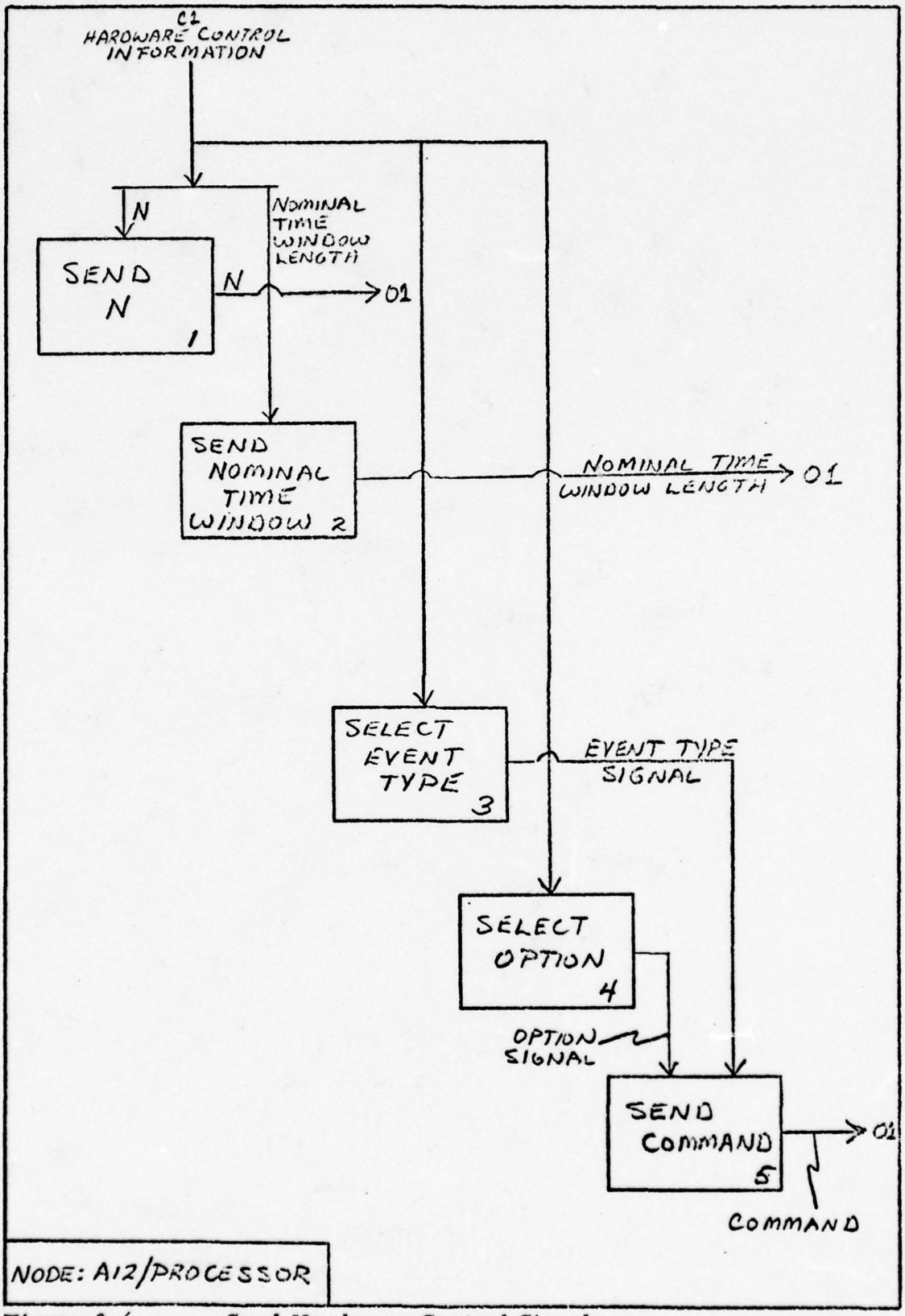


Figure 3-6 Send Hardware Control Signals

Send Hardware Control Signals (A12/Processor). The process of preparing and sending hardware control signals is specified in more detail in Figure 3-6. The control signals consist of the option parameter, either N (101) or a nominal time window length (201), the event type (301), and the selected option (401) which is pulse skip mode or nominal time window mode. The event type and option are combined into one command (5) to make the hardware operation easier.

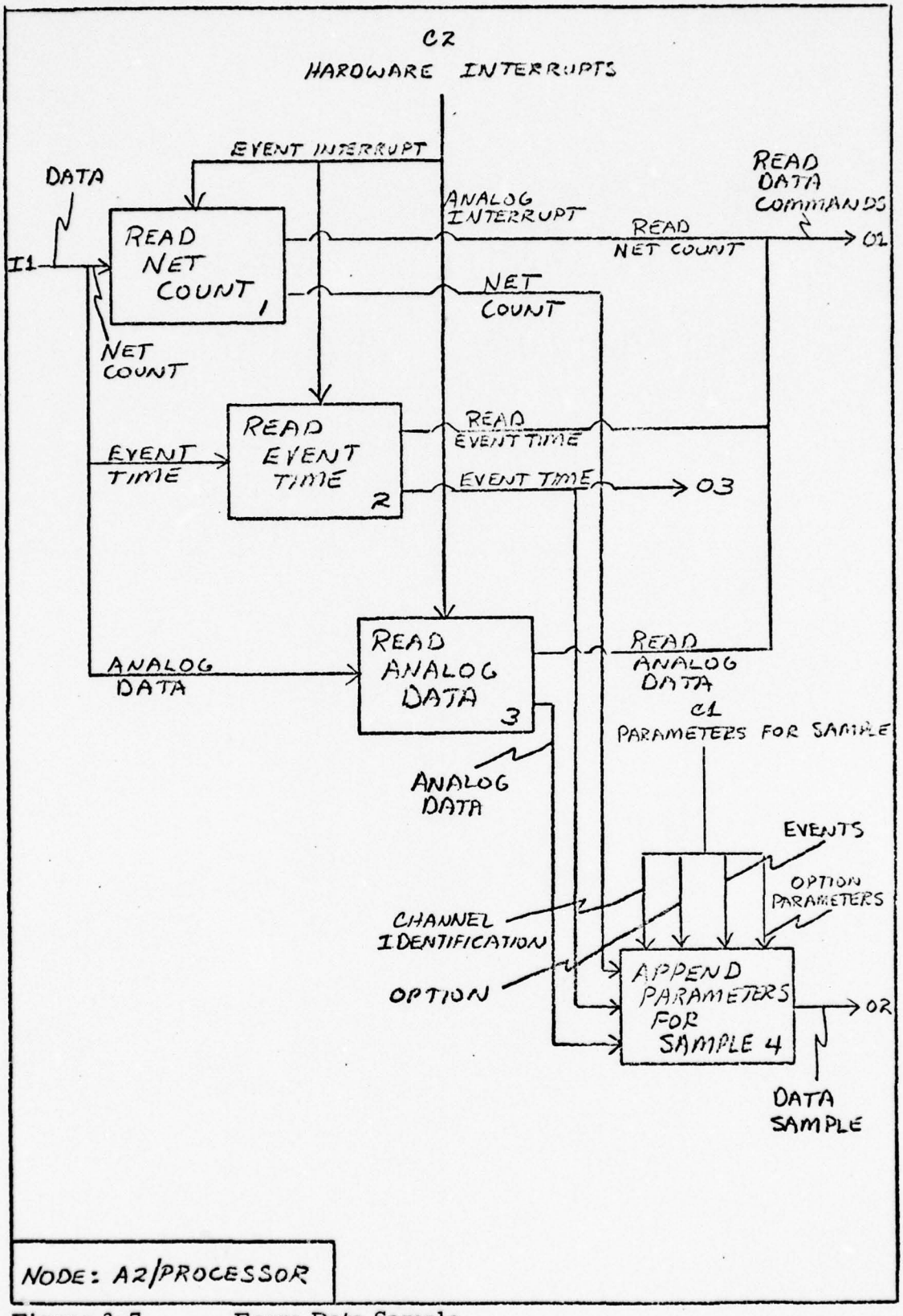


Figure 3-7 Form Data Sample

Form Data Sample (A2/Processor). Figure 3-7 shows how data samples are formed. The net count (1I1), event time (2I1), and analog data (3I1) are read from the special purpose hardware every time an event or A/D conversion causes an interrupt (C2). Parameters for the samples (C1) remain constant and can be added to a data sample (4) at any time before transmission to the minicomputer.

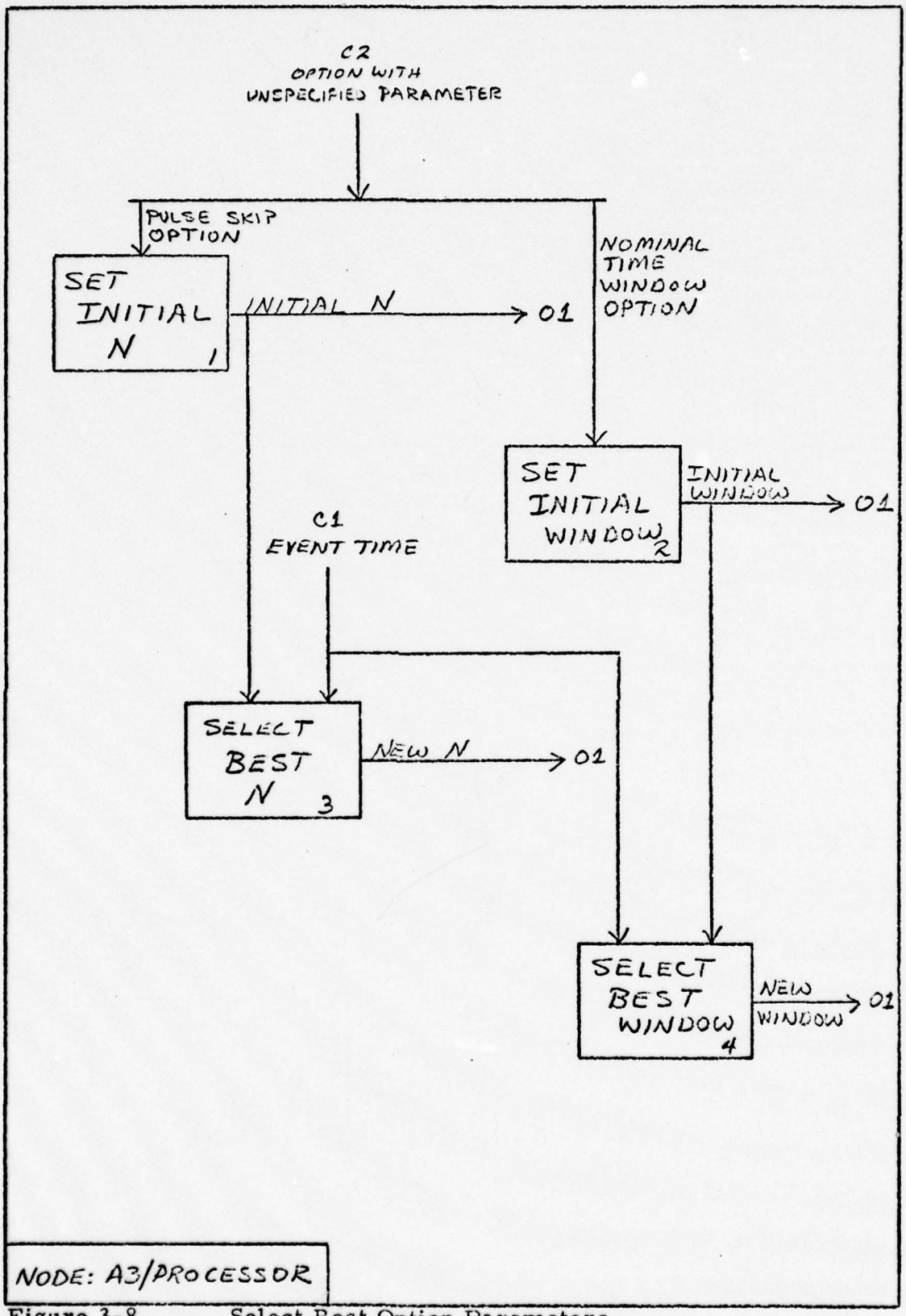


Figure 3-8 Select Best Option Parameters

Select Best Option Parameters (A3/Processor). Selecting the best option parameters, Figure 3-8, consists of establishing an initial parameter value (1 or 2), and updating the value periodically (3 or 4). The initial value may be a constant for either option. The data rate is determined from previous event times (C1) and is used to adjust the parameters to maintain the maximum continuous data rate.

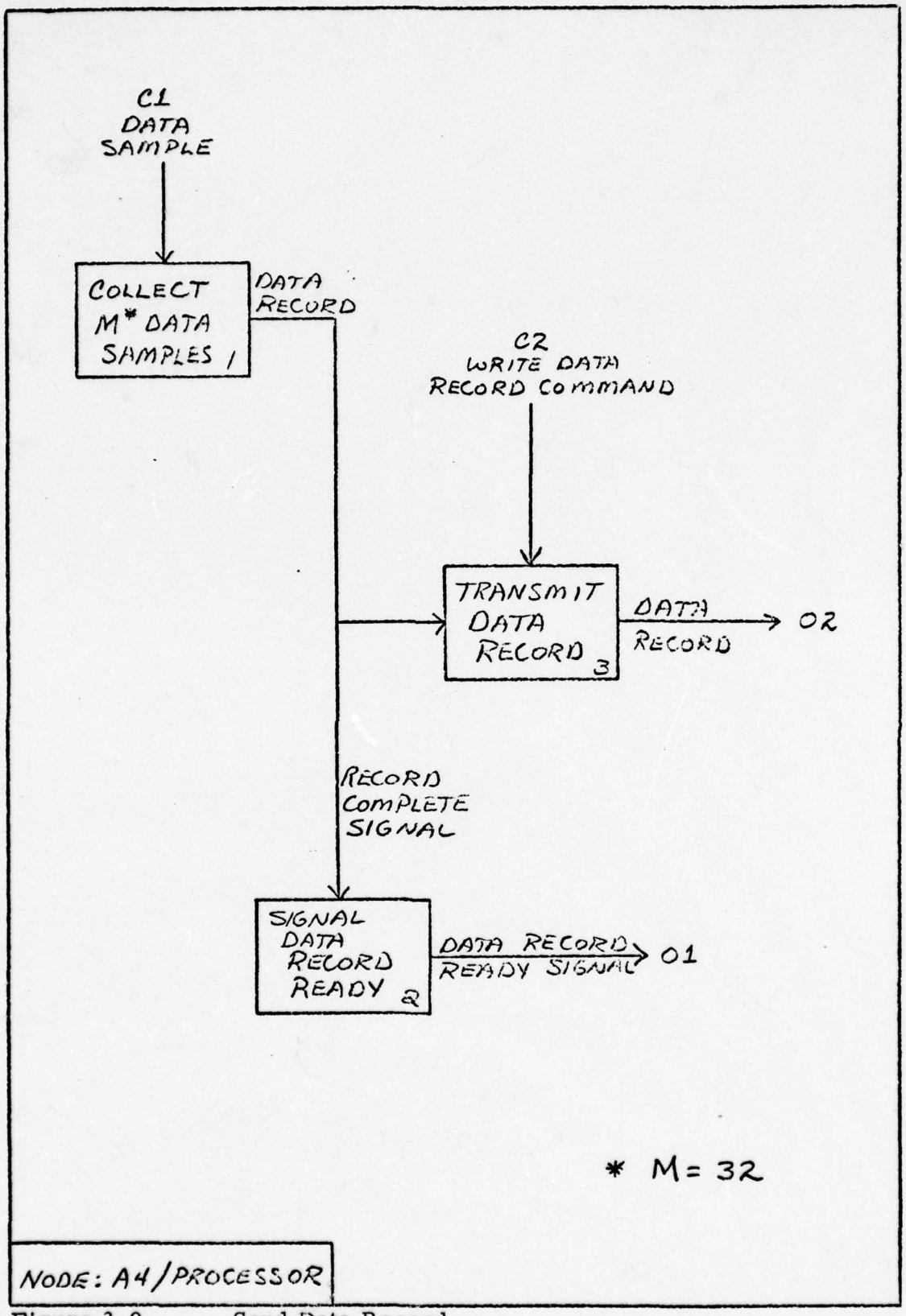


Figure 3-9 Send Data Record

Send Data Record (A4/Processor). The final node in the processor design model is Send Data Record, given in Figure 3-9. This node is identical to A45 in the requirements definition model.

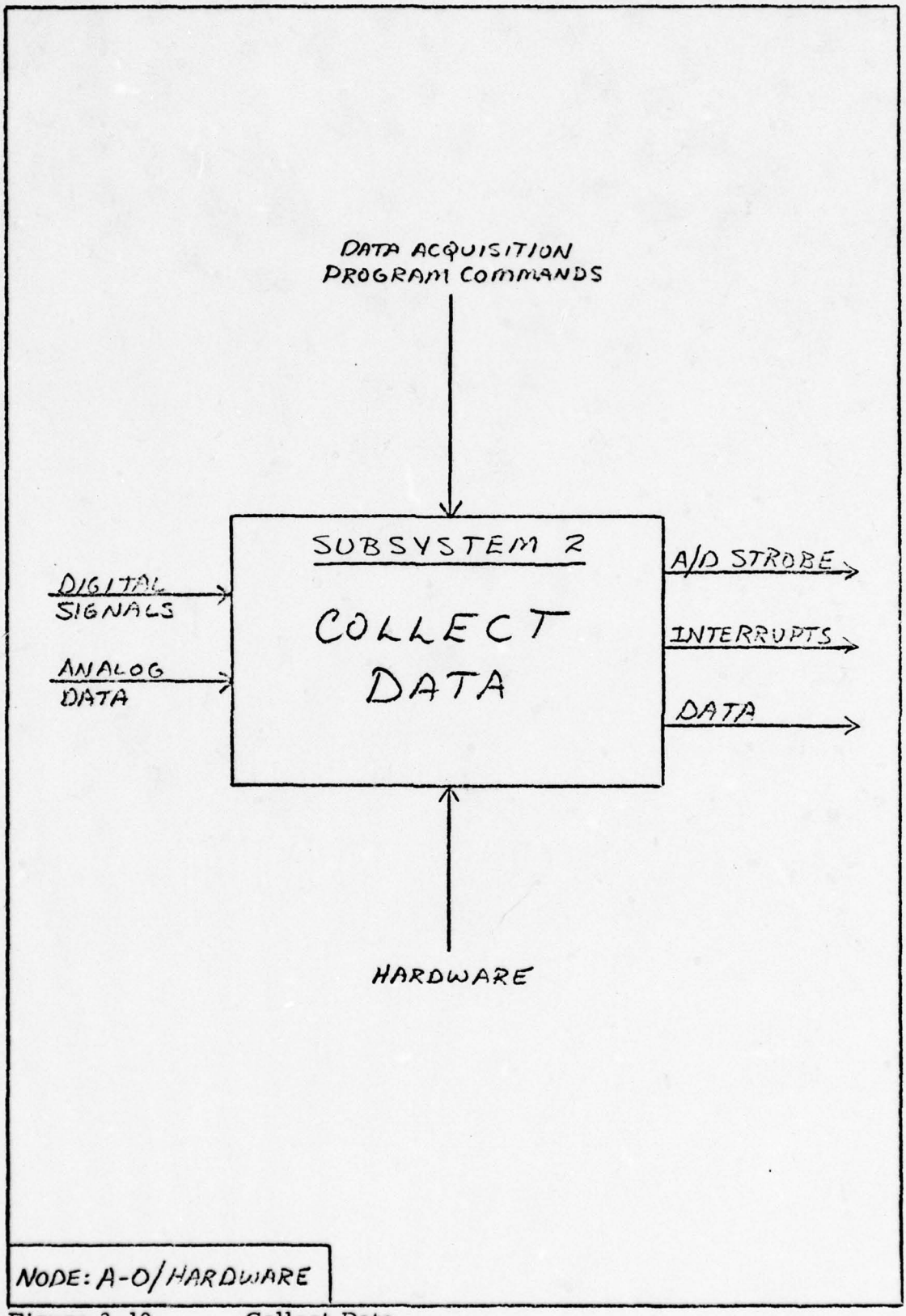


Figure 3-10 Collect Data

Subsystem 2: Collect Data (A-0/Hardware). The special purpose hardware functions are developed in the second design model, Collect Data, which begins with Figure 3-10. The viewpoint of this model is also that of the system designer. Figure 3-10 shows that the hardware must interface with the system's digital processor (C1, 02, 03) and the test instruments which monitor guidance component performance (I1, I2, 01).

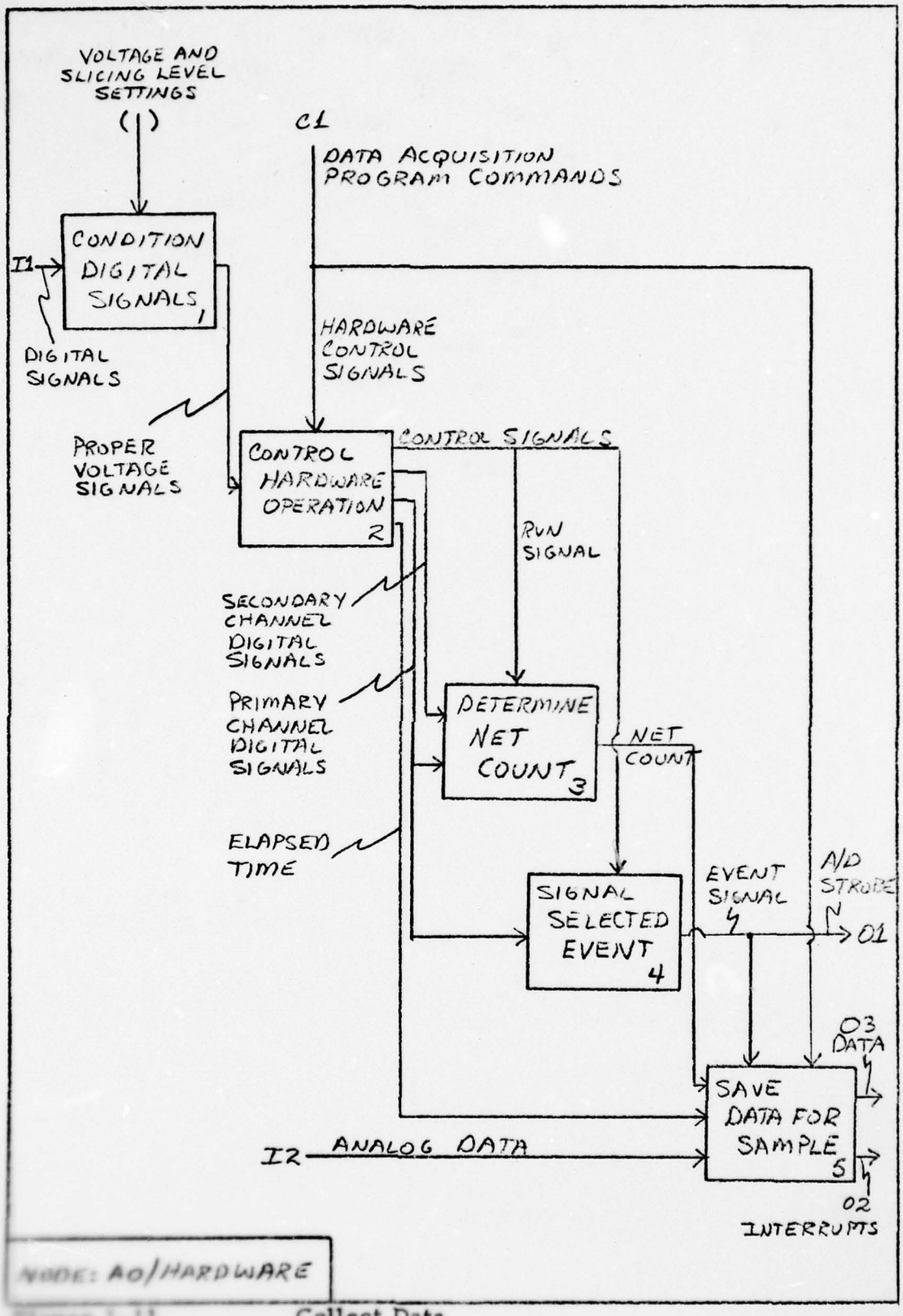


Figure 1-11 Collect Data

Collect Data (A0/Hardware). The primary special purpose hardware activities are detailed in Figure 3-11. Condition Digital Signals (1) and Determine Net Count (3) are essentially unchanged from the requirements model. Control Hardware Operation (2) is a collection of control functions which are mentioned but not diagrammed in the requirements model. Software functions are removed from Identify Selected Event (Node A3) in the original model to give Signal Selected Event (4). Finally, Save Data for Sample (5) shows the need for registers to save the event time (I3), the net count (I1), and the analog data (I2) until the values can be read by the digital processor.

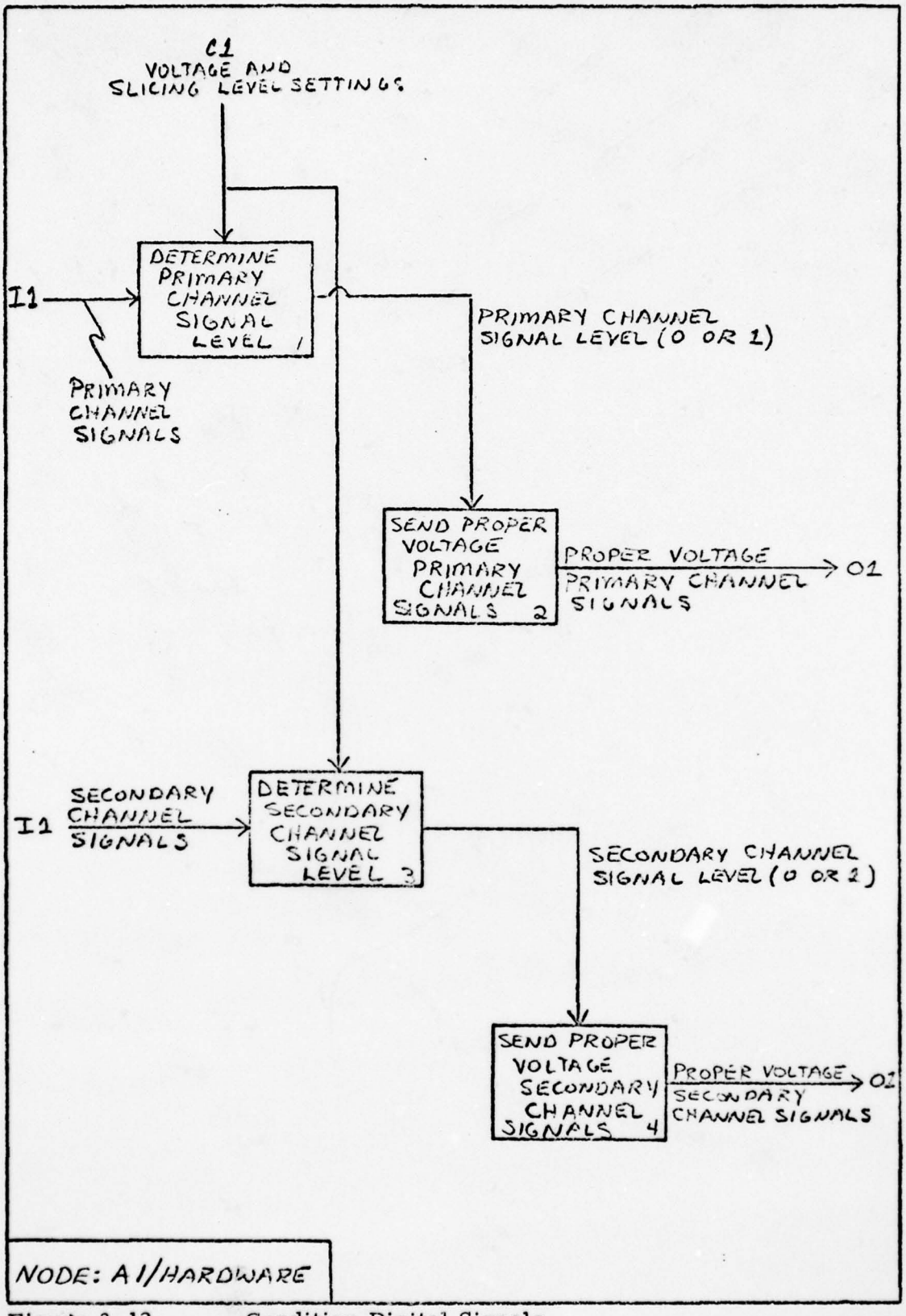


Figure 3-12 Condition Digital Signals

Condition Digital Signals (A1/Hardware). Condition Digital Signals, Figure 3-12, is unchanged from the requirements definition model. It is included here to keep the design model complete.

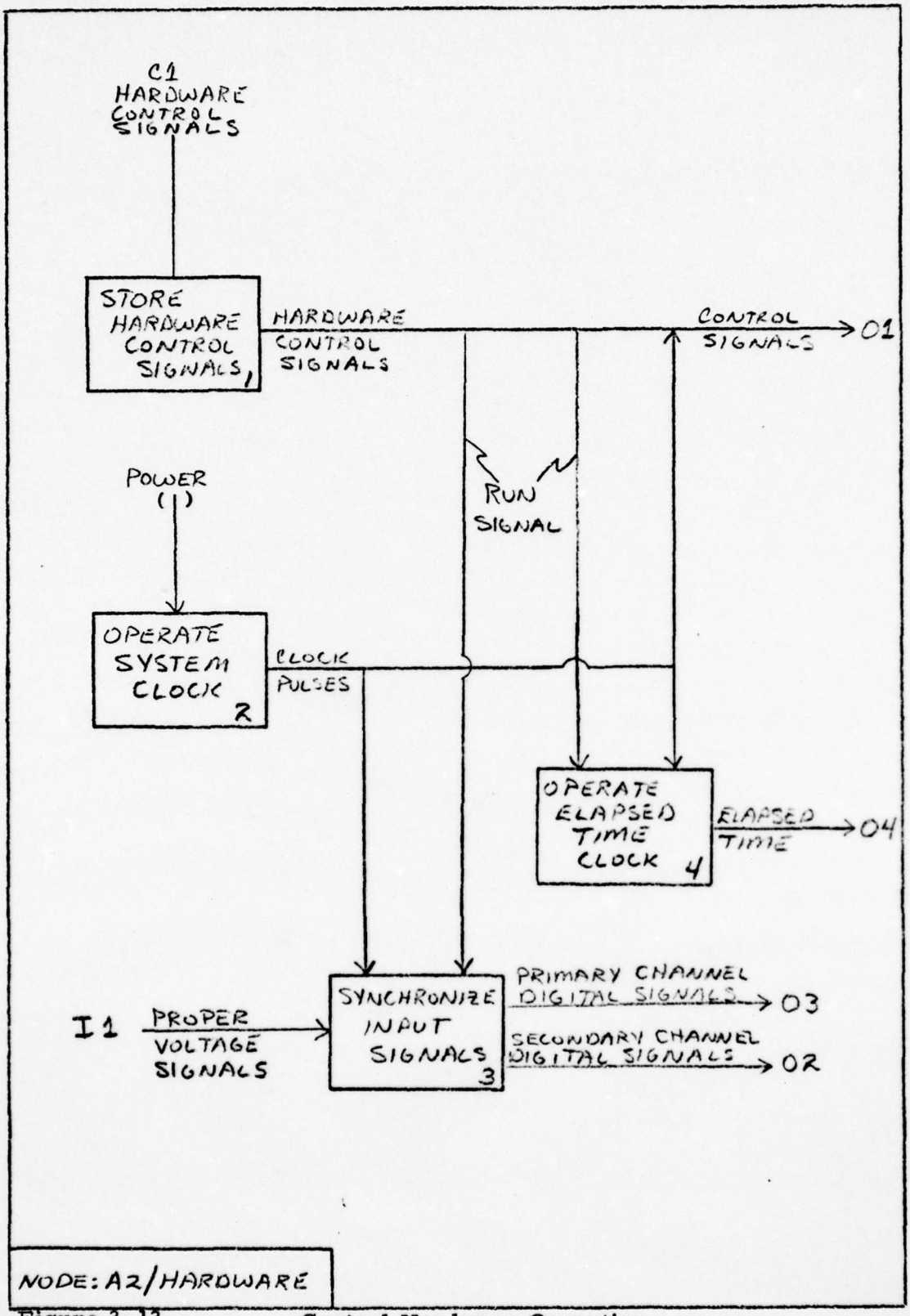


Figure 3-13 Control Hardware Operation

Control Hardware Operation (A2/Hardware). The various hardware control functions are given in Figure 3-13. Hardware control signals from the digital processor are saved and distributed appropriately. A system clock (2), which runs at 40 MHz, operates any time power is applied to the system (2C1). The input signals (I1) are synchronized with the system clock (3) so the signals can be used for other system functions. The elapsed time is also maintained (4).

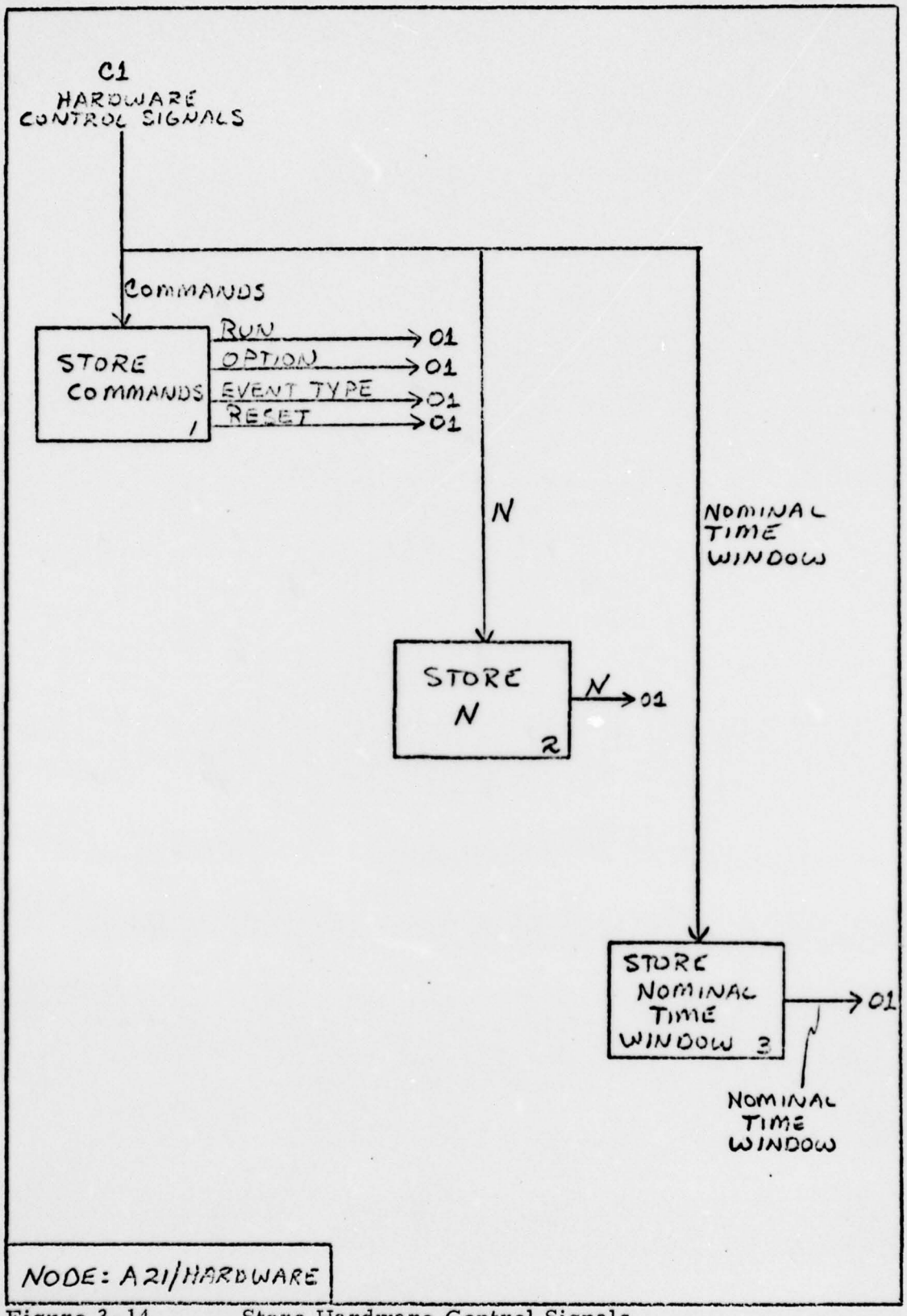


Figure 3-14 Store Hardware Control Signals

Store Hardware Control Signals (A21). The separate hardware control signals are detailed in Figure 3-14. Commands consist of a run signal (101), the selected option (102), the event type (103), and a reset signal which can be used to clear the special purpose hardware when the mode of operation is changed. Storage for N (2) and the nominal time window (3) is also provided.

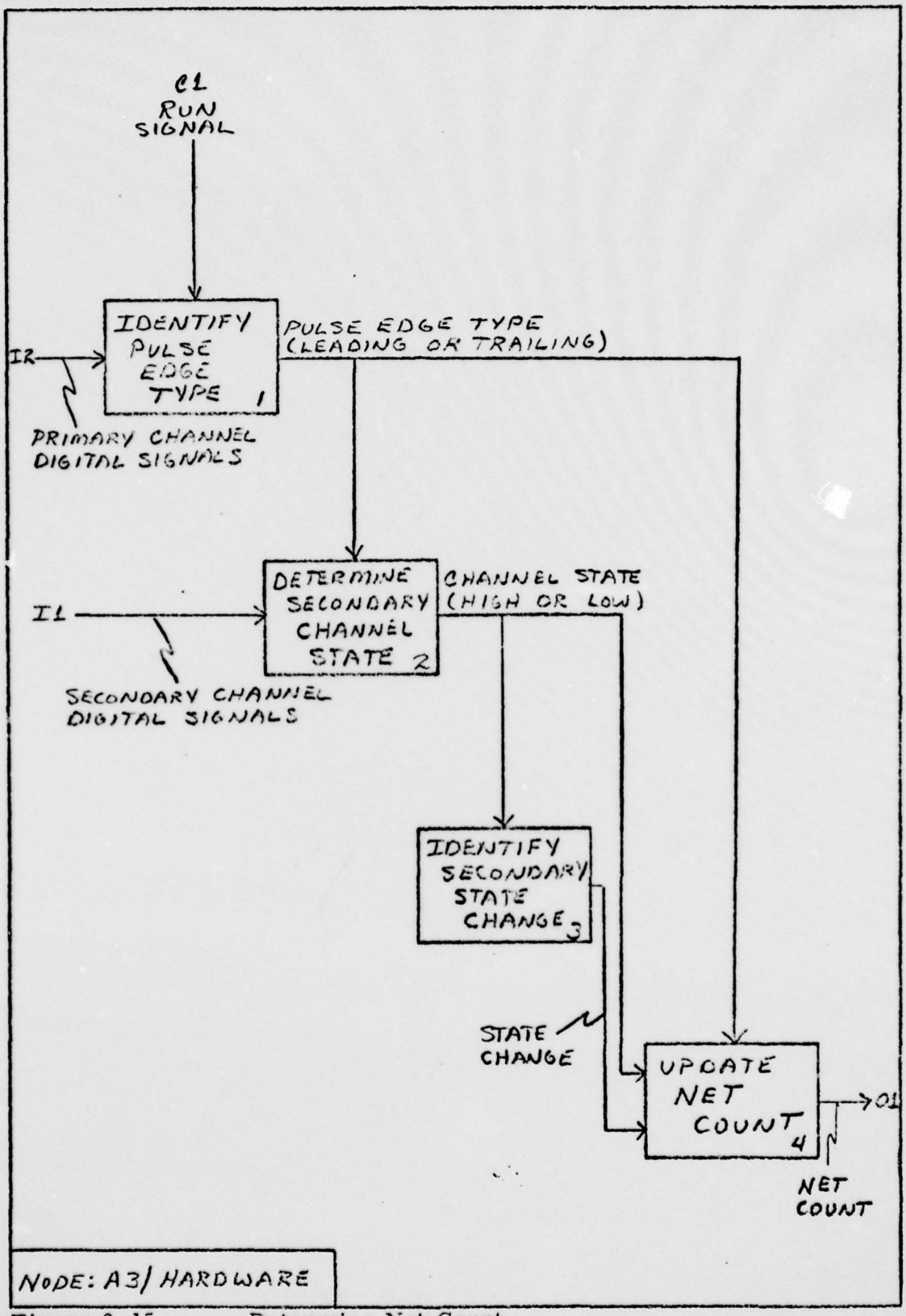


Figure 3-15 Determine Net Count

Determine Net Count (A3/Hardware). Figure 3-15 which shows the activities required for computing the net count is also repeated from the requirements definition model without change.

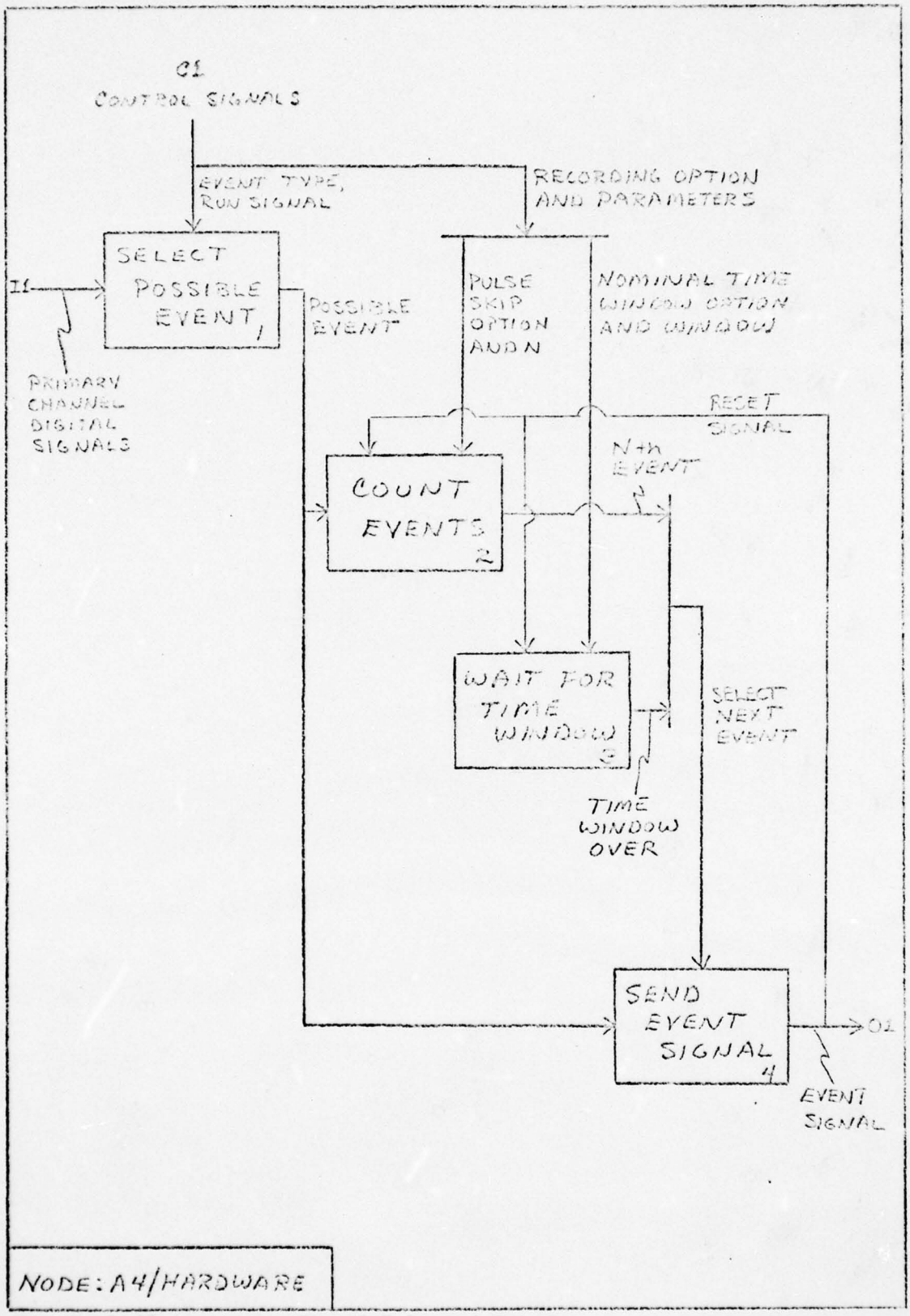


Figure 3-16 Signal Selected Event

Signal Selected Event (A4/Hardware). The hardware functions necessary for identifying and signaling events are given in Figure 3-16. This node is similar to Node A3, Figure 2-9, in the requirements definition model with the software activities removed.

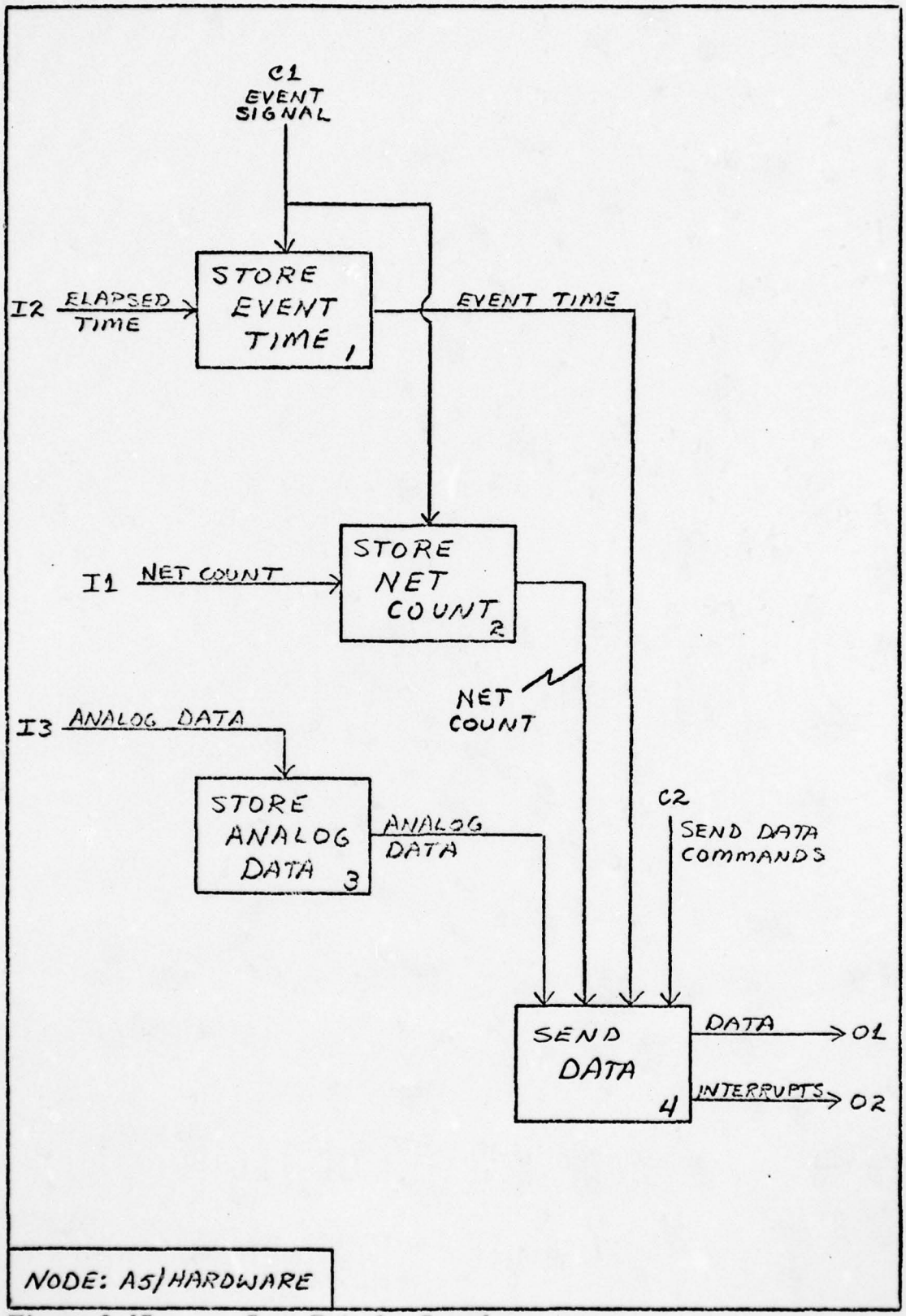


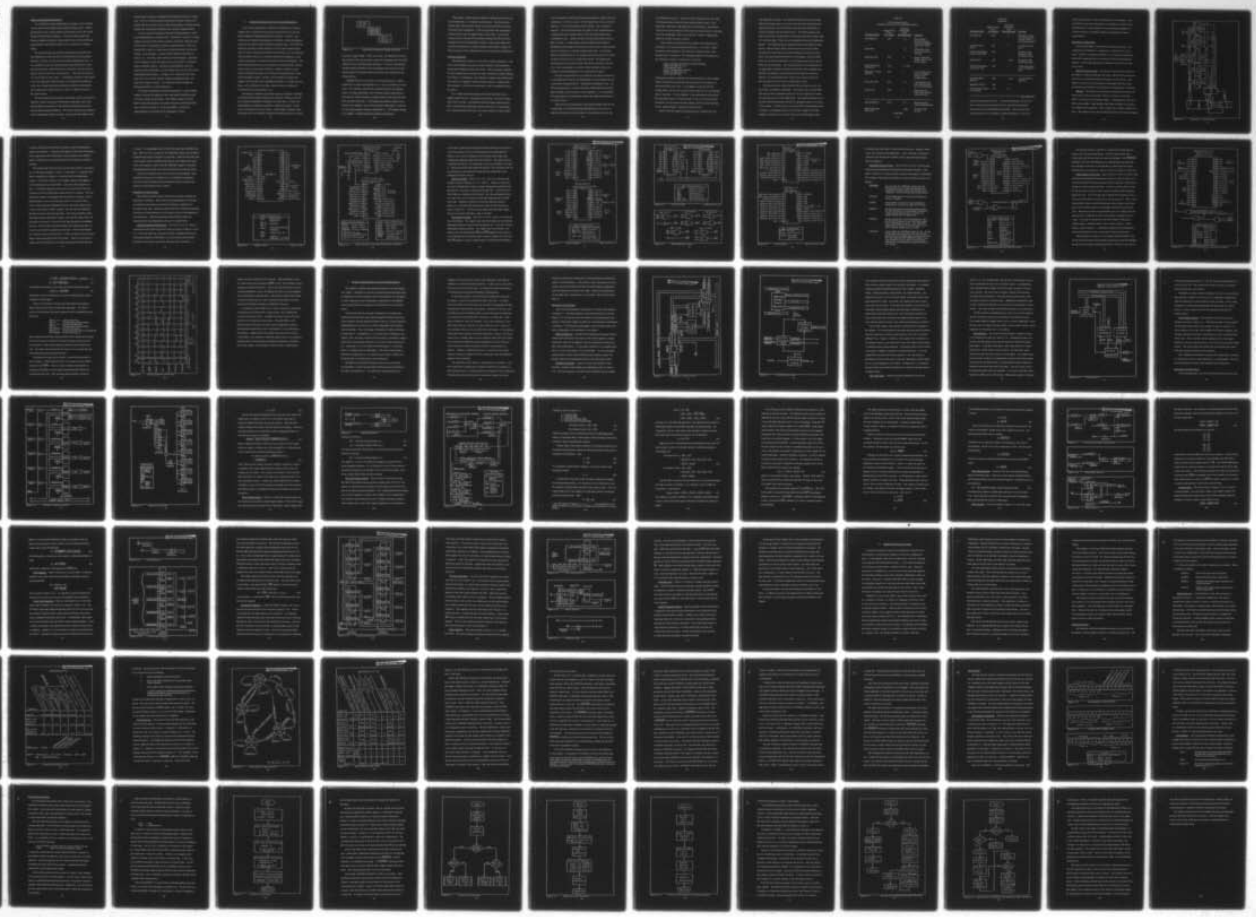
Figure 3-17 Save Data for Sample

Save Data for Sample (A5/Hardware). The final hardware activity to be modeled is saving data for a sample which is shown in Figure 3-17. The values which must be saved at the time of an event (C1) are the event time (I4) and the net count (I1). Analog data (I3) follows an event by the A/D conversion time. When data is saved, interrupts are generated (01), and data is sent to the digital processor on command (C2).

AD-A055 228 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN OF A LABORATORY DATA ACQUISITION SYSTEM (TIME DIGITIZATI--ETC(U)
MAR 78 J R MANEELY

UNCLASSIFIED AFIT/GCS/EE/78-4 NL

2 OF 3
AD
A055228



System Design Phase Observations

To conclude the system design phase discussion, a few remarks must be made about the Structured Analysis design model. Timing specifications have a strong effect on the general form of the system design and merit special attention. The reason why several new functions appeared within the design model is also worth considering along with how easily the SA nodes can be converted to a hardware layout.

The present timing specifications dictate that about half of the time digitization system functions be performed by special purpose hardware, and this can only increase system development costs. If it is possible to relax the minimum pulse width restriction to about five microseconds, determining and recording the net count can probably be done by a bit slice microprocessor. The same microprocessor can be used to count events. A substantial reduction in the system clock rate might allow other functions to be performed by software. In any case, timing restrictions determine hardware requirements, and a change in the specifications can mean a complete revision in the system design.

The new functions which appear in the design model are primarily related to control of processor and hardware operation and to data storage which is necessary for the interface between the processor and the special purpose hardware. The control activities are implicit in the requirements definition model, but must be made explicit in the

design model to provide compatibility with standard hardware components. This is a good demonstration that the SA design method can do exactly what is necessary in the system design phase; it bridges the gap between the requirements definition and a specific implementation.

In some cases the transition from the SA design model to a hardware implementation is almost trivial. Subsystem 1 provides a system software structure which makes selecting a processor relatively easy, but the direct correspondence between many Subsystem 2 nodes and standard TTL chips is a surprising and powerful benefit from the SA method. As an example, consider Control Hardware Operation in Figure 3-13. Node A21, Store Hardware Control Signals, represents a set of registers, while the system clock of node A22 is a standard feature of digital systems. Synchronize Input Signals, node A23, requires a pair of flip-flops, and the elapsed time clock of A24 can be implemented with counters. In Figure 3-15, Determine Net Count, nodes A31, A32, and A33 require simple combinational logic, and node A34 is another counter. As a hardware system design tool Structured Analysis is quite satisfactory.

To briefly summarize the system design phase, a system design method is selected and applied to the requirements definition model to create a system design model. The method is again a form of SADT because the diagrams are easily reorganized and because they show concurrent activities. In the next chapter, a processor implementation is developed for the Subsystem 1 model.

IV Hardware Selection and Circuit Layout (Subsystem 1)

In the system design phase, hardware and software are treated together and in much the same manner because the two are just different methods of implementing logical functions. However, once hardware and software requirements are defined, system development must proceed along two interrelated but separate lines. In this thesis, hardware selection includes selection of all IC chips and the preparation of a block diagram which shows how the chips are to be organized. Thus, hardware selection is related to hardware development as the system design phase is related to the entire system development. The block diagrams which come from hardware selection provide a framework to direct circuit layout activities. Software structure is the analog to hardware selection in the software development line; it preceeds and guides coding. For this reason hardware selection and software structure have been identified separately in a single life cycle phase, and are followed by another compound phase consisting of circuit layout and coding (Figure 4-1).

The pairing of hardware and software design activities is intended to show that while design techniques are different, hardware and software cannot be developed in isolation from each other. In fact, the interdependence of hardware selection, software structure, circuit layout, and coding is all very strong. In this design, hardware selection begins first, but software structure determines the choice of some

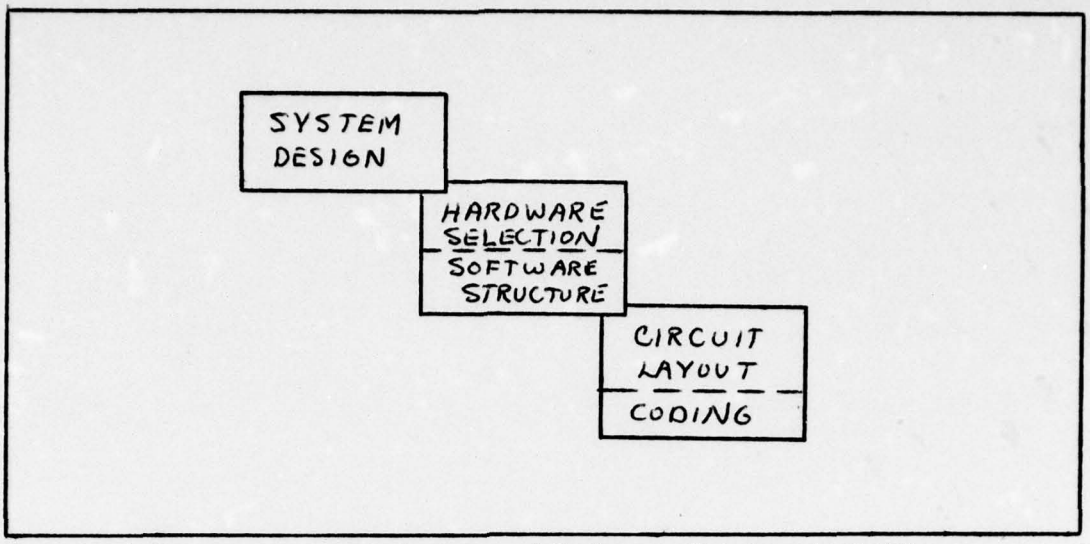


Figure 4-1 Hardware and Software Design Activities

processor support chips. While some code, I/O addresses for example, depend on the circuit layout, instruction limitations also force the addition of more special purpose hardware. It is fair to say that the iteration between hardware selection/software structure and circuit layout/coding is far greater than for any other pair of phases in the design process.

Although iteration is necessary in the design process, a paper which traces a design development as it occurs is not likely to be very clear. It is natural to present the complete hardware and software parts of a design separately, and to maintain logical continuity it is necessary to combine hardware selection with circuit layout and software structure with code. In this paper the hardware design is presented in two chapters, one for the processor (Subsystem 1), and one for the special purpose hardware (Subsystem 2). The software design is in Chapter VI which follows the hardware presentation.

This chapter, which describes hardware selection and circuit layout for Subsystem 1, is divided into three parts. The first section explains the criteria used to select a processor and lists the processors which were considered. In the second section the organization of the processor and its support chips is described. A part of the second section is devoted to the interface between the HP 2100 minicomputer and the time digitization system, a topic which affects the processor hardware and a software protocol described in Chapter VI. The last section of this chapter gives the circuit layout for Subsystem 1.

Processor Selection

Essentially two criteria are used here to select a processor. The first is the capability to perform all the functions in the Subsystem 1 design model within the given time limits. After functional capability the next consideration is system life cycle cost which includes expenses for hardware, development, and maintenance (repair and modification). The criteria are applied to three processor options: a single board minicomputer, a bit slice microprocessor, and a conventional microprocessor.

For a single board minicomputer the initial hardware cost is slightly higher than for a microprocessor system, but basic development costs are lower. One difficulty with the single board minicomputer is that the time digitization system must have a 16 bit parallel bus connection with the HP 2100, and it also requires a large number

of I/O connections with the special purpose hardware (eight 12 bit A/D converters, a 16 bit net count, a 36 bit elapsed time count, an 8 bit N register, a 32 bit nominal time window register, and a command register). The I/O requirements are likely to force modifications to a single board minicomputer which would raise the hardware and development costs. Another problem with the single board minicomputer is speed. A minicomputer with an eight bit word length is probably not fast enough to transmit two words every 16 microseconds which is necessary to meet the maximum data rate. For reasons which are explained below, an average instruction time of 2.67 microseconds is necessary for an eight bit processor, but single board minicomputers are normally slower than this to keep their cost low. If a 16 bit minicomputer is selected, initial hardware costs go up enough to justify development of a microprocessor system.

Bit slice microprocessors can easily give the speed necessary in a time digitization system, but development costs are higher than for conventional microprocessors. Beyond the development costs, a bit slice microprocessor is unique and more complicated than any other type of system so maintenance is more expensive. If a conventional microprocessor can be found which meets the speed requirements, it is a better choice.

The processing requirements in the system design model are not complicated so assuming the timing specifications can be met, an eight bit conventional microprocessor is the best processor for the

time digitization system. Because of their widespread use, the eight bit microprocessors have the lowest initial hardware cost. More people have experience with eight bit microprocessors, and development aids are more common for this type of microprocessor so development and modification costs can be kept low. Finally, replacement parts are cheaper and more common.

It now remains to determine if any eight bit conventional microprocessors can meet the timing specifications. To do this, a short benchmark code segment for the most time critical process, data transmission is necessary. The following sequence of instructions is the minimum necessary to transmit a 16 bit word:

```
Load a register from memory (using indexing)
Output one eight bit word
Increment the index
Load a register from memory
Output one eight bit word
Increment the index
```

Repeating the sequence explicitly 16 times transmits one data sample without resorting to instructions which keep track of the number of words which have been sent. An assumption is that the HP 2100 accepts data as fast as the microprocessor sends it so no acknowledgement is required. To be acceptable, a microprocessor must execute the benchmark code in 16 microseconds or less (2.67 microseconds/instruction, 256 microseconds/data sample, 8192 microseconds/data record, approximately 1 second/122 data records).

Table II lists the eight bit microprocessors considered for the

time digitization system. The instruction execution time associated with each microprocessor is in most cases the time required for the fastest instruction, but the minimum execution time still allows an initial screening of the microprocessors. The bottom group in the table contains the microprocessors with instruction times greater than 2.67 microseconds. These are not acceptable for the time digitization system. The middle group has execution times which might be satisfactory, however these contain integral read only memory (ROM). Because only a few copies of the time digitization system are anticipated, any mask programmed ROM is unacceptable, and since ROM needs are uncertain at this point, any microprocessor with on-board ROM may be undesirable. The microprocessors in the top group are all apparently acceptable in instruction execution time although more detailed consideration of each instruction set might be necessary. However, in the top group other criteria can be used to select the best microprocessor for the time digitization system.

The characteristics which separate the top group are cost, support chips, and general availability. The Zilog Z-80 is an Intel 8080 type device with an extended instruction set and a higher price which are both unnecessary in this application. The eight bit address line on the RCA CDP 1802 promises to slow I/O operations which are critical in this system. The Electronic Arrays EA 9002 is interesting because it contains a scratchpad memory of 64 words, enough for two data samples, but the lack of a second source and limited support chips

Table III

8 Bit Microprocessors
Considered for the Time Digitization System

Microprocessor	Approximate Price (in volume)	Minimum Instruction Time (Microseconds)	Remarks
Intel 8080A-1	$20	1.3	Most popular microprocessor; best set of support chips (Ref 12:86)
Intel 8085	-	1.3	8080A plus clock and bus control on one chip
Motorola 6800	$30	2	Fewer chips than 8080A; one $\pm$ 5 volt power supply
MOS Technology 6502/6512	$20	2.5 typ.	
Electronic Arrays EA 9002	$20	2	12 bit address bus; 64x8 scratchpad memory; no second source
RCA CDP 1802	$24	2.5	8 bit address line with external latch for 16 bit addresses
Zilog Z-80	$50	2	8080A type machine with extended instruction set
Intel 8048/8078	$10	2.5	Mask programmable ROM/EPROM
MOS Technology MCS 660X	-	2 typ.	Up to 2K ROM on chip

(continued)

Table III
(Continued)

Microprocessor	Approximate Price (in volume)	Minimum Instruction Time (Microseconds)	Remarks
Fairchild F8	$10	2	1K ROM on chip; slow I/O (4 microseconds CPU; 8 microseconds PSU)
Fairchild 3859 (F8-1)	$13	2	One chip version of F8
General Instrument Corp. Series 8000	$22	-	European; predecessor of F8; 1K ROM on chip
Mostek 3870	$10	2 typ.	F8 system with 2K ROM on chip
General Instrument Corp. PIC 1650	$20	1	12 bit instruction; 512 word ROM on chip
Rockwell PPS-8 PPS-8/2	$24	4 typ.	1 or 2K ROM on PPS-8/2
Signetics 2650A-1	$20	6	
National Semiconductor, SC/MP	$18	10	

(Ref 11:45-73)

keep it from being first choice. The MOS Technology 6502 and 6512 have a typical instruction speed close to the maximum allowable which makes their acceptability questionable. All of the top three microprocessors are acceptable, but the Intel 8080A-1 is the first

choice here because it is the most familiar to this designer. The Intel 8085 has the same advantages as the 8080A-1 and offers some simplifications, but its integrated interrupt handling capability is not quite enough to completely support time digitization system requirements.

Subsystem 1 Organization

With the Intel 8080A-1 selected as the system processor, the organization of Subsystem 1 components is relatively easy to diagram. Figure 4-2 presents the 8080A-1 in a standard configuration along with components that are necessary for the time digitization system. A brief description of the components is given in the next few sections.

Basic Processor Group. The basic processor group consists of the 8080A-1 Central Processing Unit (CPU), the 8224 Clock Generator and Driver, and the 8828 System Controller and Bus Driver. The 8080A-1 processes all the programs, and together the three chips provide the clocking and control signals for the Subsystem 1 hardware.

Memory. Any number of memories are satisfactory, however an access time of 320 nanoseconds or less is necessary to allow the 8080A-1 to operate at its maximum speed. Looking ahead to the software coding chapter, approximately 2048 bytes of ROM are necessary. For demonstration purposes, a pair of Intel 3628-4 bipolar PROMs are used. The random access memory (RAM) needed for the system depends

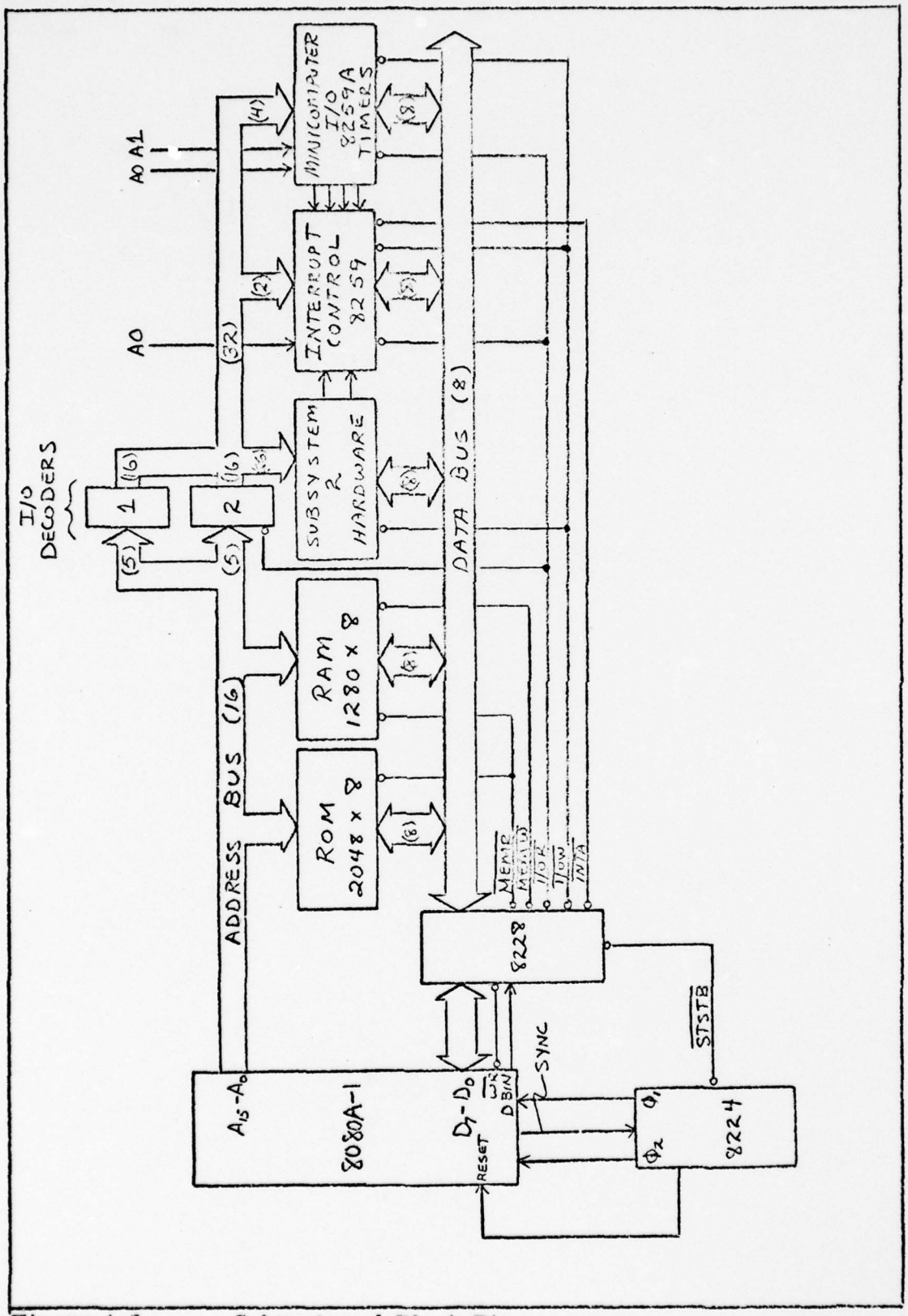


Figure 4-2 Subsystem 1 Block Diagram

on how many data records are to be accumulated at one time. A reasonable minimum is two data records, one complete and waiting for transmission and one being filled. In this design, enough storage is provided for the minimum of two records in addition to the memory required for proper software operation. Ten Intel 2111A-2 Static RAM chips (each 256x4) are used here to give 1280 bytes of data and program storage. Additional memory to hold more data records would give more flexibility for programming the HP 2100, but the extra hardware raises the system price.

I/O Decoders. Two 4-line to 16-line decoders (Texas Instruments or TI SN 74154) are used to address I/O registers located in Subsystem 2, the interrupt control chip, and the minicomputer I/O chip.

Subsystem 2 Hardware. Registers in the Subsystem 2 hardware are connected to the bidirectional data bus and are addressed by 26 lines from the two I/O decoders. Subsystem 2 is discussed in detail in the next chapter.

Interrupt Control. Interrupt control is provided by an Intel 8259, Programmable Interrupt Controller (PIC). Six interrupt lines are used, two from Subsystem 2 and four from the minicomputer I/O hardware, and the purpose of each line is explained later. The 8259 provides priority among the interrupt request lines and allows interrupts to be masked when necessary. When the 8080A-1 acknowledges an interrupt, the PIC provides a subroutine call instruction (not the usual restart instruction) which begins the interrupt service routine.

Minicomputer I/O. An Intel 8255A, Programmable Peripheral Interface (PPI), is used to create the 16 bit parallel data bus between the HP 2100 and the time digitization system. The PPI contains three 8 bit I/O ports, two of which are connected to the HP 2100 I/O bus. The 8080A-1 reads or writes to the ports one at a time while the HP 2100 communicates with both ports in one operation. A two stage timer works along with the 8255A in this configuration to allow the 8080A-1 to transmit data to the HP 2100 without waiting for an acknowledgement of each word. If the HP 2100 fails to accept a word within the necessary time, a timer runs out and interrupts the 8080A-1, stopping data transfer until the HP 2100 is ready. As a number of methods for interfacing the HP 2100 and the time digitization system are possible, and the selected interface has a substantial impact on both hardware and software, this subject needs more thorough consideration.

Minicomputer - Time Digitization System Interface. The most efficient, and most expensive, interface method available for the HP 2100 is direct memory access (DMA). This method would do the most toward reducing HP 2100 processing requirements for data acquisition. However, if simpler interface methods still allow enough time for data analysis, then the less expensive methods may be more desirable.

Hewlett Packard sells several interface boards for the HP 2100 which can be used with a time digitization system. Custom designs which make full use of the potential of the 8255A are also possible, but in this design the least complex of the standard HP 2100 interface boards

is used to determine the functions necessary in the time digitization system I/O hardware. Using the least expensive HP 2100 I/O hardware does complicate the time digitization system hardware and software which is another factor to consider when the final interface method is selected.

The standard HP 2100 interface board for this design is assumed to be a 16 Bit Duplex Register, 12554A, or equivalent. A separate board with an individual I/O address is used with each time digitization system. Each board contains a 16 bit buffer register and supplies a 16 bit bidirectional bus to the 8255A. Only two control signals are used. A CONTROL flip-flop on the board is set by the HP 2100 and indicates to the time digitization system that I/O is enabled. The time digitization system acknowledges HP 2100 actions by setting a FLAG flip-flop which is also on the interface board. For the HP 2100 to write a word to the time digitization system, the minicomputer loads a word in the interface buffer register, and then sets the CONTROL flip-flop and clears the FLAG flip-flop. The rising CONTROL signal interrupts the time digitization system, and when the system has read both bytes of the word, it sets the FLAG flip-flop which acknowledges the data. The HP 2100 may loop until the FLAG signal goes high, or it may use the FLAG to generate an interrupt. For the time digitization system to write to the HP 2100, it loads two bytes into the interface buffer register through the 8255A I/O ports. When the second byte is loaded, the FLAG flip-flop is set which signals the HP 2100 that data

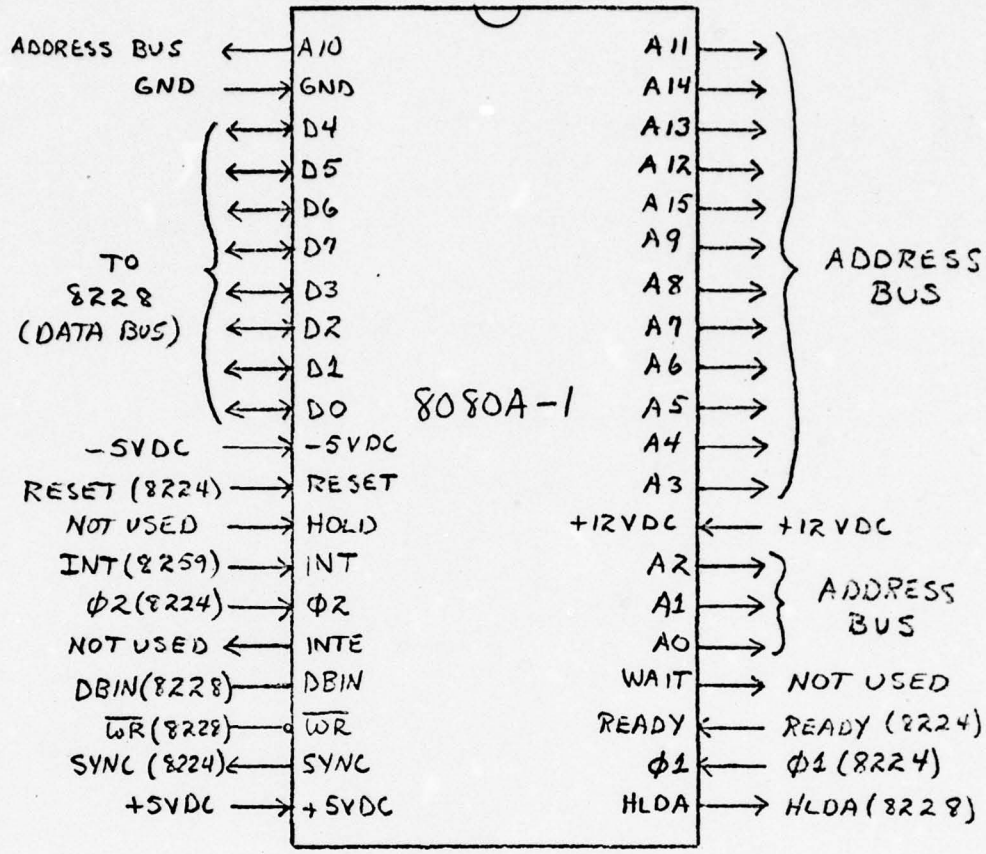
is ready. To acknowledge data, the HP 2100 resets the CONTROL flip-flop. When no I/O is in progress, the CONTROL signal must be high to enable the flag signal to generate an interrupt. Hence the HP 2100 must clear and then set the CONTROL flip-flop for each output word; input to the minicomputer requires that the CONTROL signal be cleared to acknowledge data and then set to allow transmission of the next word.

Because the interface board uses only two control signals, some ambiguity may arise as to which device is initiating I/O operation. The cause of the ambiguity and the techniques used to overcome the problem are described in later sections.

Subsystem 1 Circuit Layout

This section presents the pin connections and logic equations for Subsystem 1 hardware. Much of the wiring information for each chip is contained in the figures and only the important connections are discussed in the text. References which describe in detail the operation and electrical characteristics of each chip are given at the beginning of each subsection. Only the operational characteristics which are important to the time digitization system are repeated here.

Basic Processor Group Circuits. (Ref 13:10.11-10.46) The pin connections for the basic processor group are given in Figures 4-3 and 4-4. The three chips are connected normally, although memory synchronization signals for the 8080A-1 are not used because the ROM and RAM access times must be less than the clock cycle time. A



PIN FUNCTIONS	
INT	INTERRUPT REQUEST
Φ1, Φ2	CLOCKS
SYNC	SYNCHRONIZATION SIGNAL
DBIN	DATA BUS IN
$\overline{WR}$	WRITE - MEMORY OR I/O

Figure 4-3 8080A-1 CPU (Ref 13:10.12)

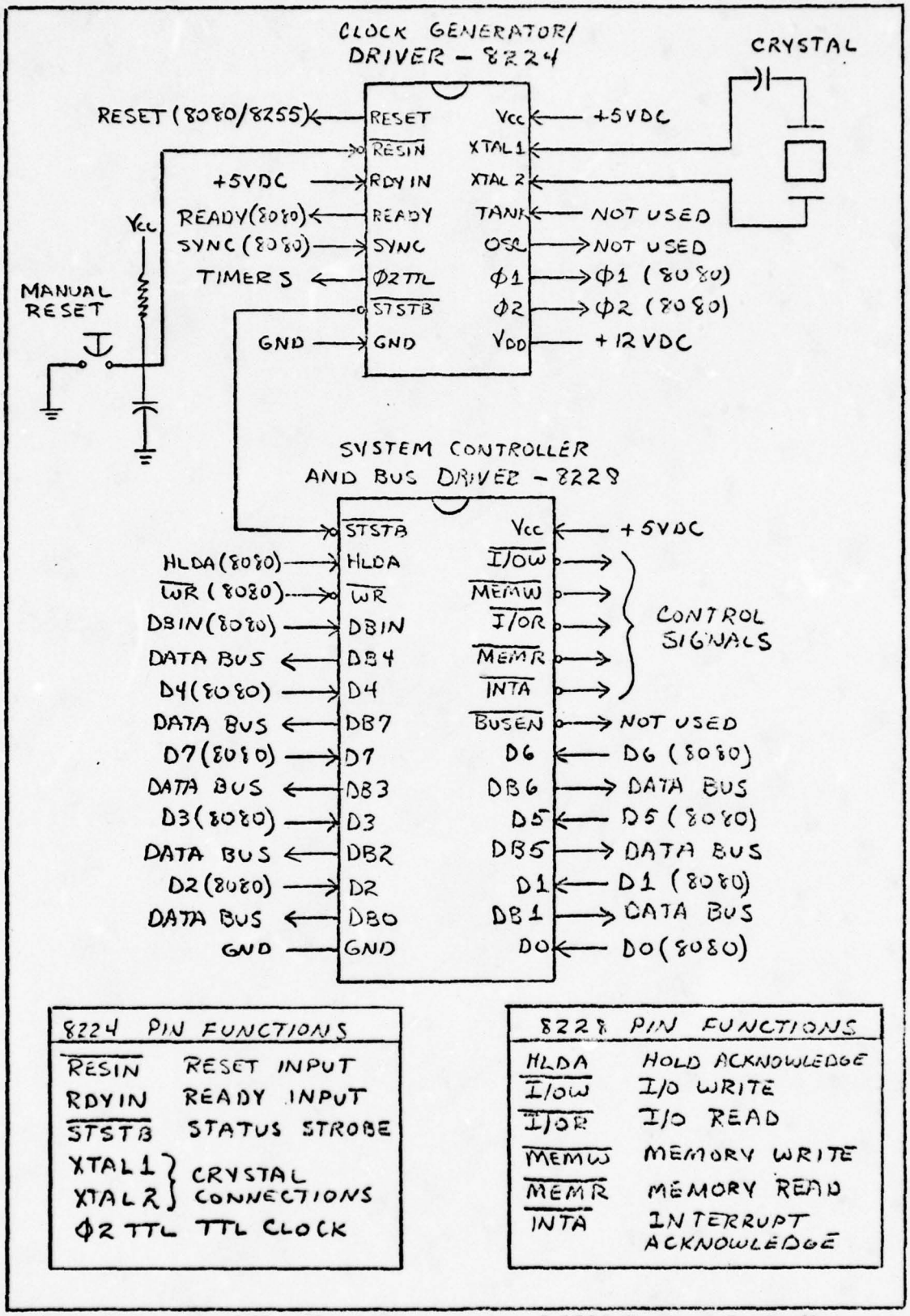


Figure 4-4 8224 Clock Generator and 8228 System Controller

28.125 MHz crystal would provide the fastest instruction cycle time of .32 microseconds, but it is recommended for this design that the 8080A-1 clock cycle be adjusted so the maximum data output rate matches the maximum input rate of the HP 2100. The reason for this is that the HP 2100 can not accept data as fast as the 8080A-1 can send it. When the HP 2100 falls behind the 8080A-1, the I/O timers will run out and cause an interrupt which stops the data transmission and ultimately slows the I/O rate.

Memory Circuits. (Ref 13:3.51-3.53, 2.68-2.71) Memory chip wiring is shown in Figures 4-5, 4-6, and 4-7. Figure 4-5 gives the two 3628-4 PROMS where chip A contains hexadecimal addresses 000 through 3FF, and chip B contains 400 through 7FF. The ten 2111A-2 RAM chips are organized into five pairs, one chip containing the four low order bits of a byte and the second chip containing the four high order bits. Standard connections for all the chips are shown in Figure 4-6, and individual chip selection logic for the five pairs is given in Figure 4-7. All the memory chips have three state outputs which can be active only when a chip is selected.

I/O Decoder Circuits. (Ref 14:7.171-7.174) Figure 4-8 shows the two I/O decoders. The output lines from the decoders go low when the appropriate address is present on the address bus. Line A4 is used to distinguish between decoders. The output lines from decoder 1 are gated at the next chip with the $\overline{\text{I/OR}}$ and $\overline{\text{I/OW}}$ signals as necessary to avoid activating an I/O register when a memory address is on the bus. The $\overline{\text{I/OR}}$ signal is used to enable decoder 2 because all the registers

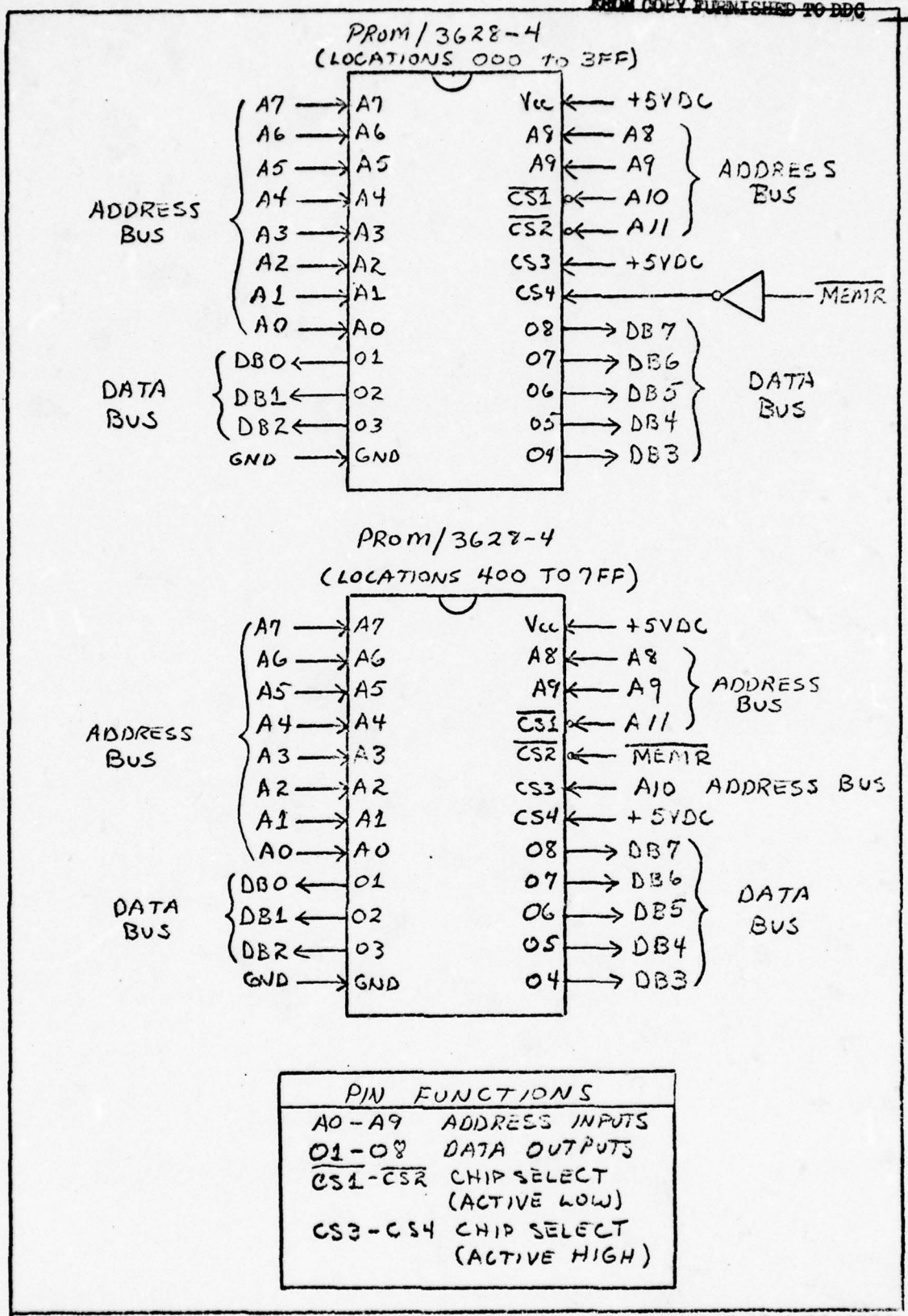


Figure 4-5 3628-4 PROMs (Ref 13:3.51-3.54)

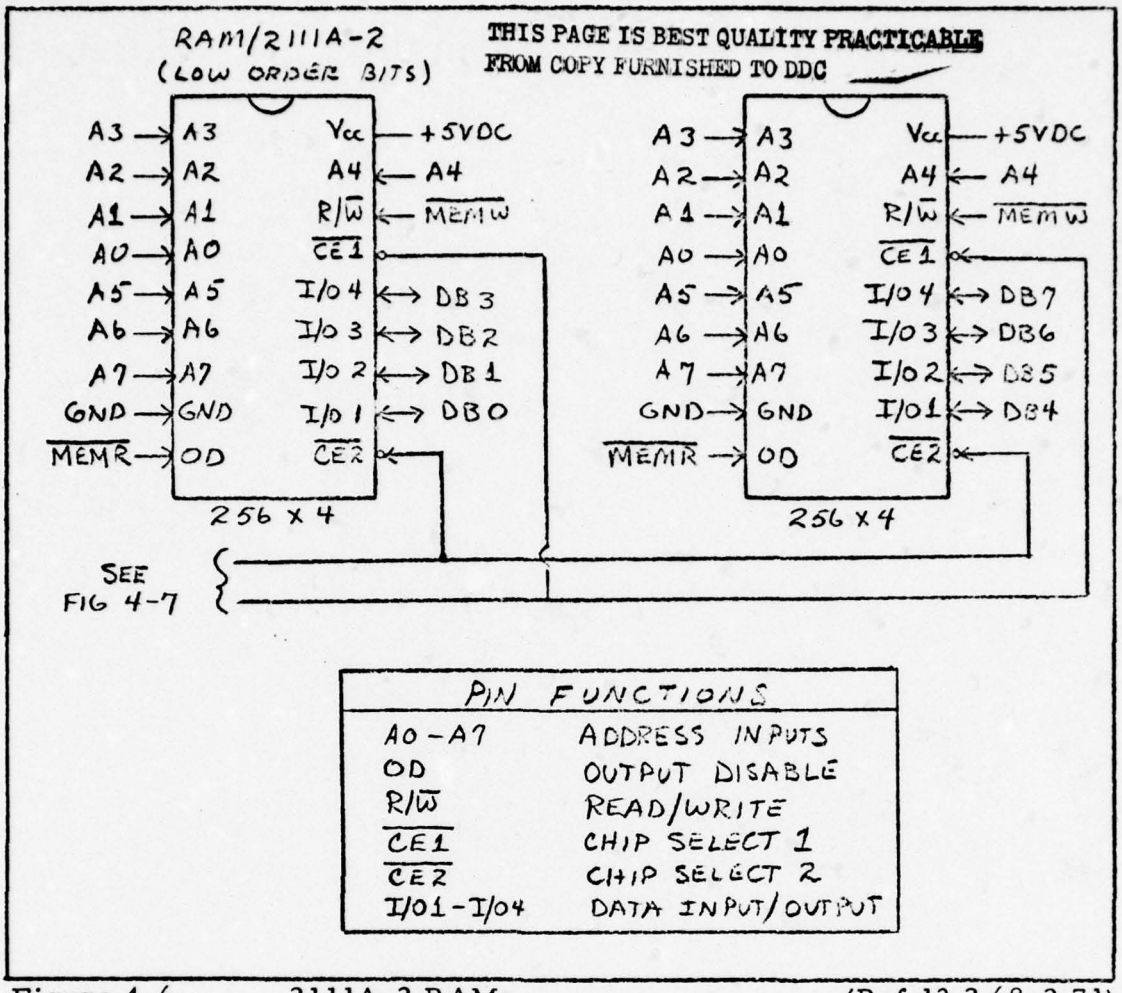


Figure 4-6 2111A-2 RAMs (Ref 13:2.68-2.71)

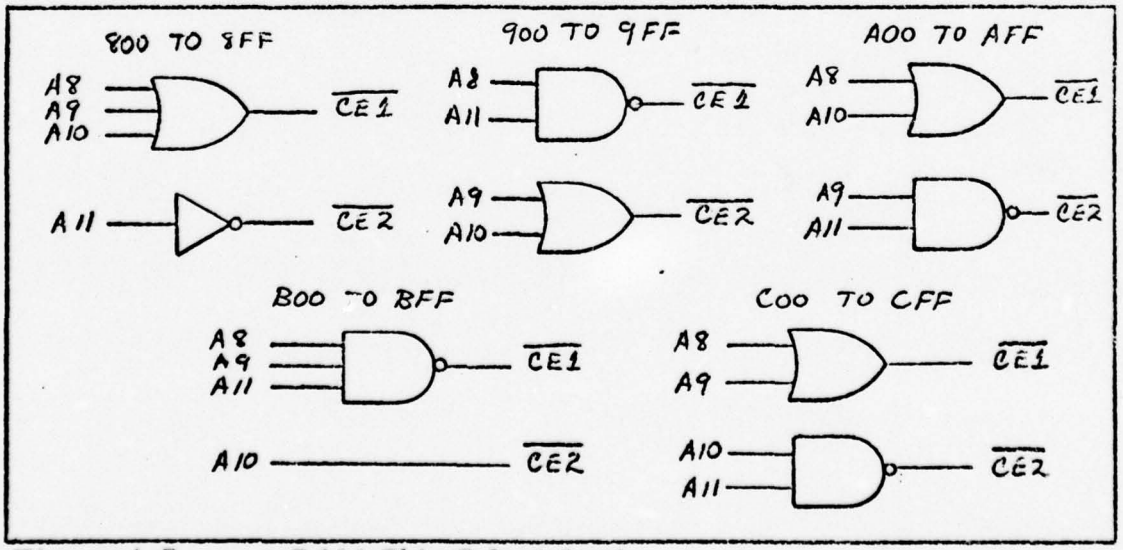


Figure 4-7 RAM Chip Select Logic

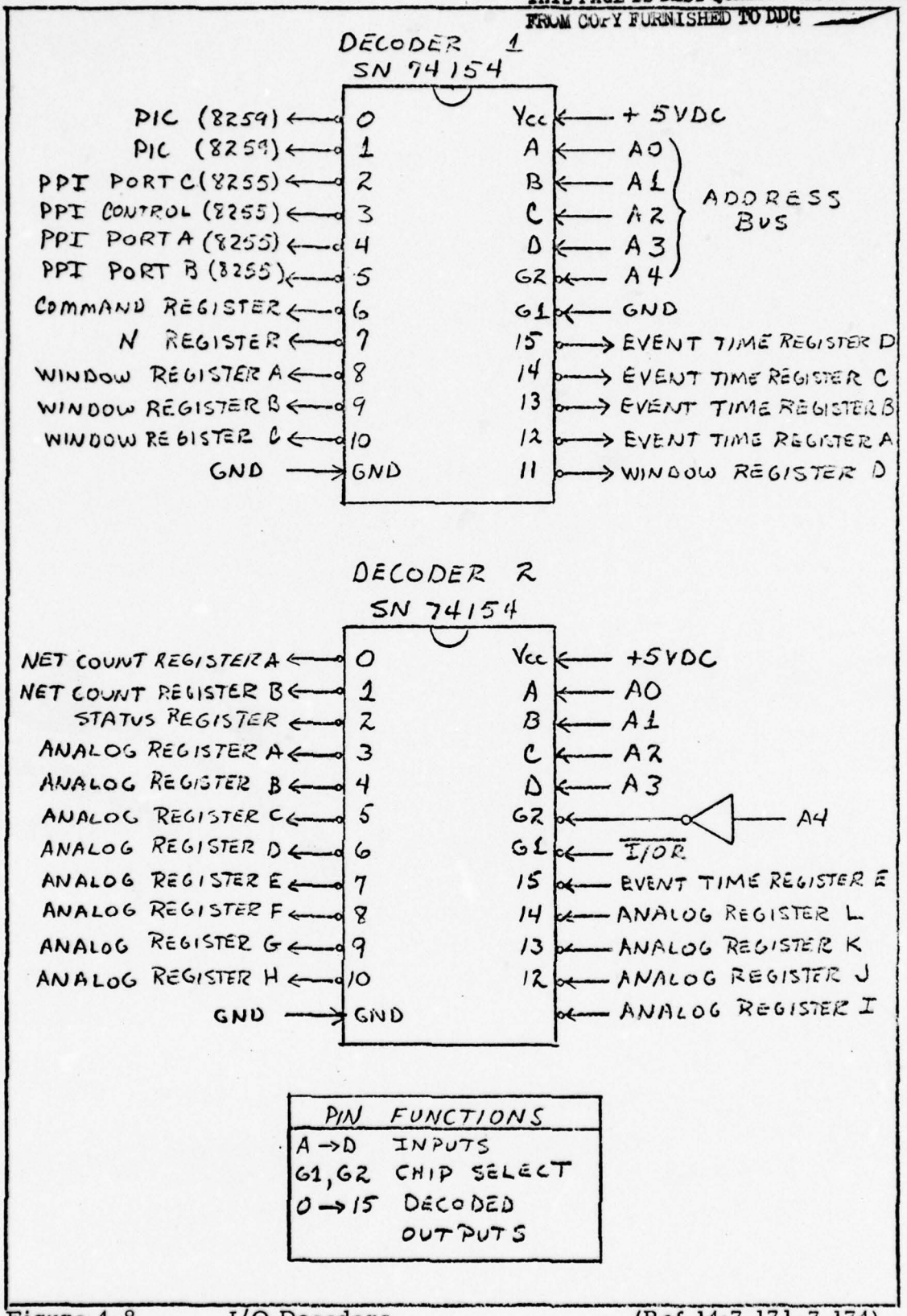
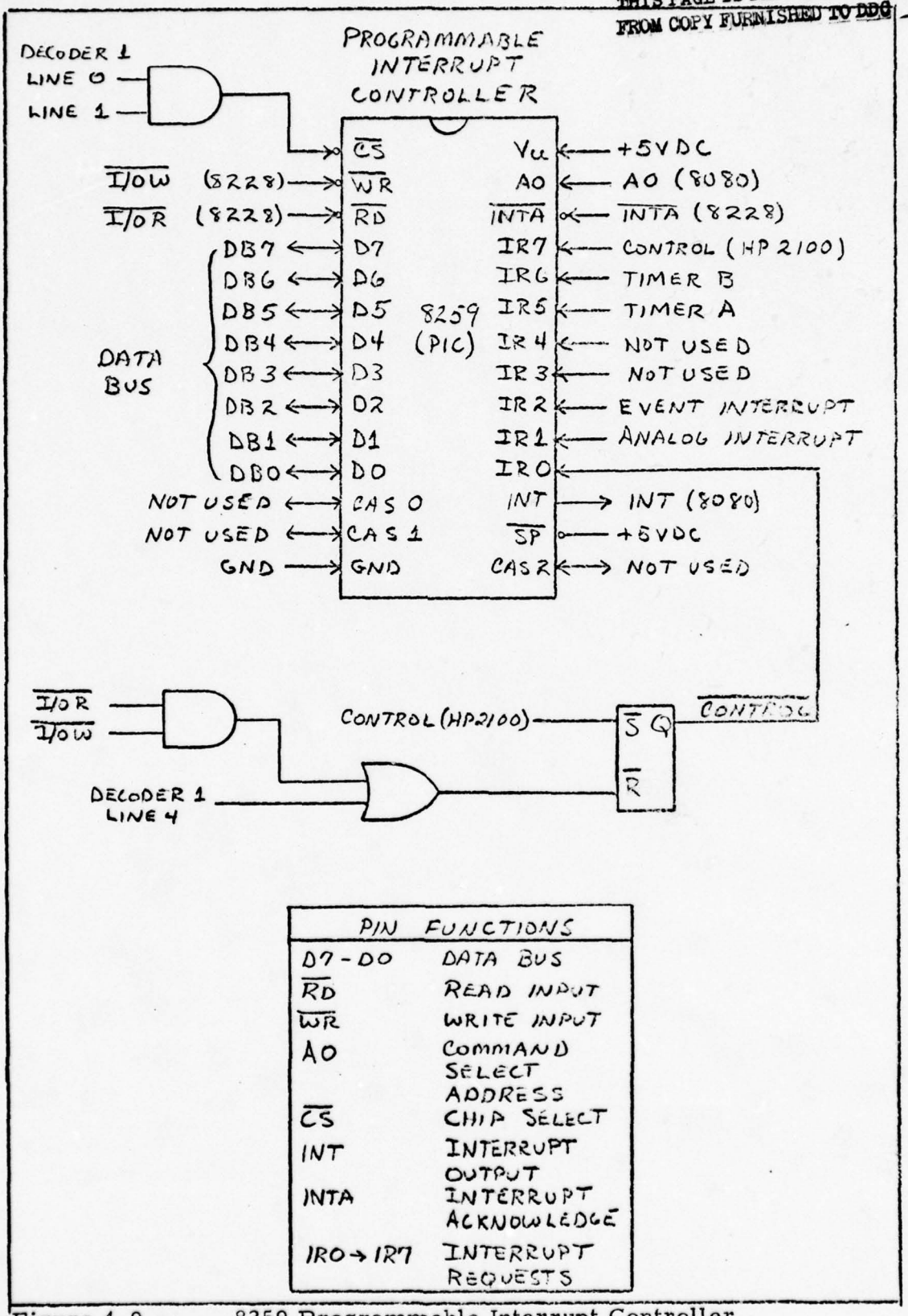


Figure 4-8 I/O Decoders (Ref 14:7.171-7.174)

it selects store data which is read into the processor. Register names are for the most part self explanitory. Lines 0 through 5 of decoder 1 reflect the fact that the PIC and PPI contain separately addressable internal registers.

Interrupt Control Circuits. (Ref 13:10.212-10.227) All interrupt requests pass through the Programmable Interrupt Controller, 8259, shown in Figure 4-9, where priority among the interrupts is established. The ranking of interrupts from highest to lowest and their purpose is as follows:

$\overline{\text{CONTROL}}$	Occurs when the CONTROL signal from the HP 2100 goes low and indicates that the time digitization system cannot interrupt the minicomputer. (not used during data transmission)
ANALOG	Occurs when the A/D conversion following an event is complete.
EVENT	Occurs when an event has been identified in Subsystem 2, and data is ready to be stored.
TIMER A	Occurs during data record transmission when the HP 2100 does not acknowledge reception of a word before the next word is ready to be loaded into the 8255A output ports.
TIMER B	Occurs during data record transmission when the HP 2100 does not set the CONTROL signal after acknowledging a word. The CONTROL signal must be set so the FLAG signal can be set to indicate another word is ready.
CONTROL	Occurs when the CONTROL signal is set. Except during data record transmission, the I/O protocol used here provides for a command from the HP 2100 each time the CONTROL signal is set. This method is used to allow communication between the minicomputer and the time digitization system.



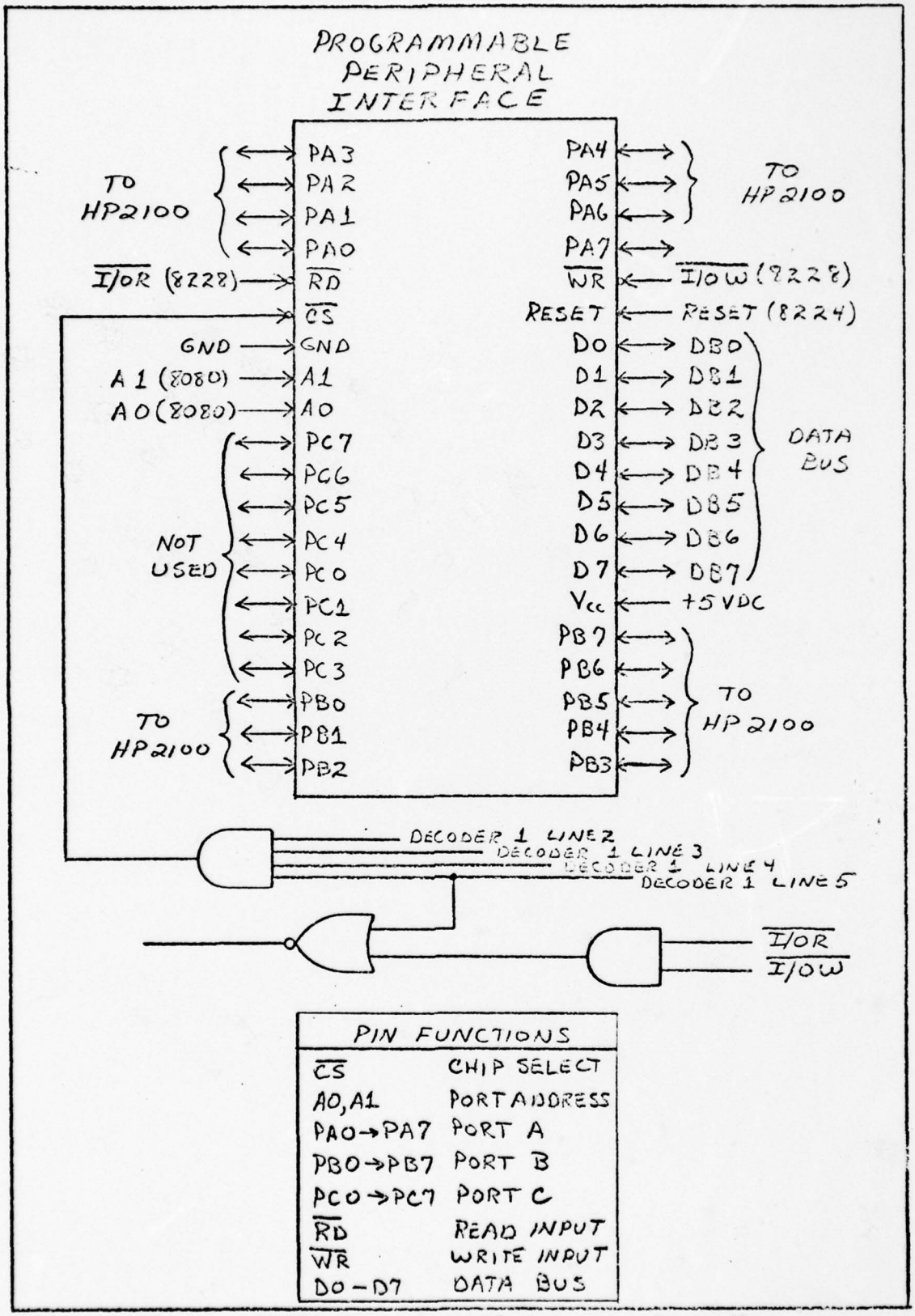
PIN	FUNCTIONS
D7 - D0	DATA BUS
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0	COMMAND SELECT ADDRESS
$\overline{CS}$	CHIP SELECT
INT	INTERRUPT OUTPUT
INTA	INTERRUPT ACKNOWLEDGE
IR0 → IR7	INTERRUPT REQUESTS

Figure 4-9 8259 Programmable Interrupt Controller

An interrupt request in the PIC is caused by the rising edge of a signal on any interrupt request line. The interrupt request must remain high until the interrupt has been acknowledged. The $\overline{\text{CONTROL}}$ interrupt is the only line which does not automatically stay high until the interrupt is serviced so a SR flip-flop is used to hold the signal. The flip-flop can be reset by a read or a write to I/O port A in the PPI.

Minicomputer I/O Circuits. (Refs 13:10.170-10.191; 14:7.489-7.495) The 8255A Programmable Peripheral Interface is given in Figure 4-10. The PPI provides the interface between the 8 bit data bus of the 8080A-1 and the 16 bit bus from the HP 2100. I/O port A (PA0 to PA7) is used for the low order 8 bits of a word, and port B (PB0 to PB7) holds the high order bits. The FLAG flip-flop in the HP 2100 interface board is set any time the 8080A-1 reads from or writes to port B. Thus, in the code presented later, the time digitization system always reads from port A first and then port B to acknowledge commands from the HP 2100. When data records are transmitted, port A is loaded first; then when port B is loaded the FLAG is set to signal a complete word is ready. The mode of operation for the PPI is Mode 0 which gives latched outputs, unlatched inputs, and no control signals. In this design, control signals for coordination between the time digitization system and the HP 2100 are used by the I/O timers and the PIC.

The I/O timers in Figure 4-11 are used to keep the 8080A-1 and the HP 2100 synchronized during data transmission without the 8080A-1 having to rely on status checking instructions. Timer A, the first of



PIN FUNCTIONS	
$\overline{CS}$	CHIP SELECT
A0, A1	PORT ADDRESS
PA0 → PA7	PORT A
PB0 → PB7	PORT B
PC0 → PC7	PORT C
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
D0 – D7	DATA BUS

Figure 4-10 8255A Programmable Interrupt Controller

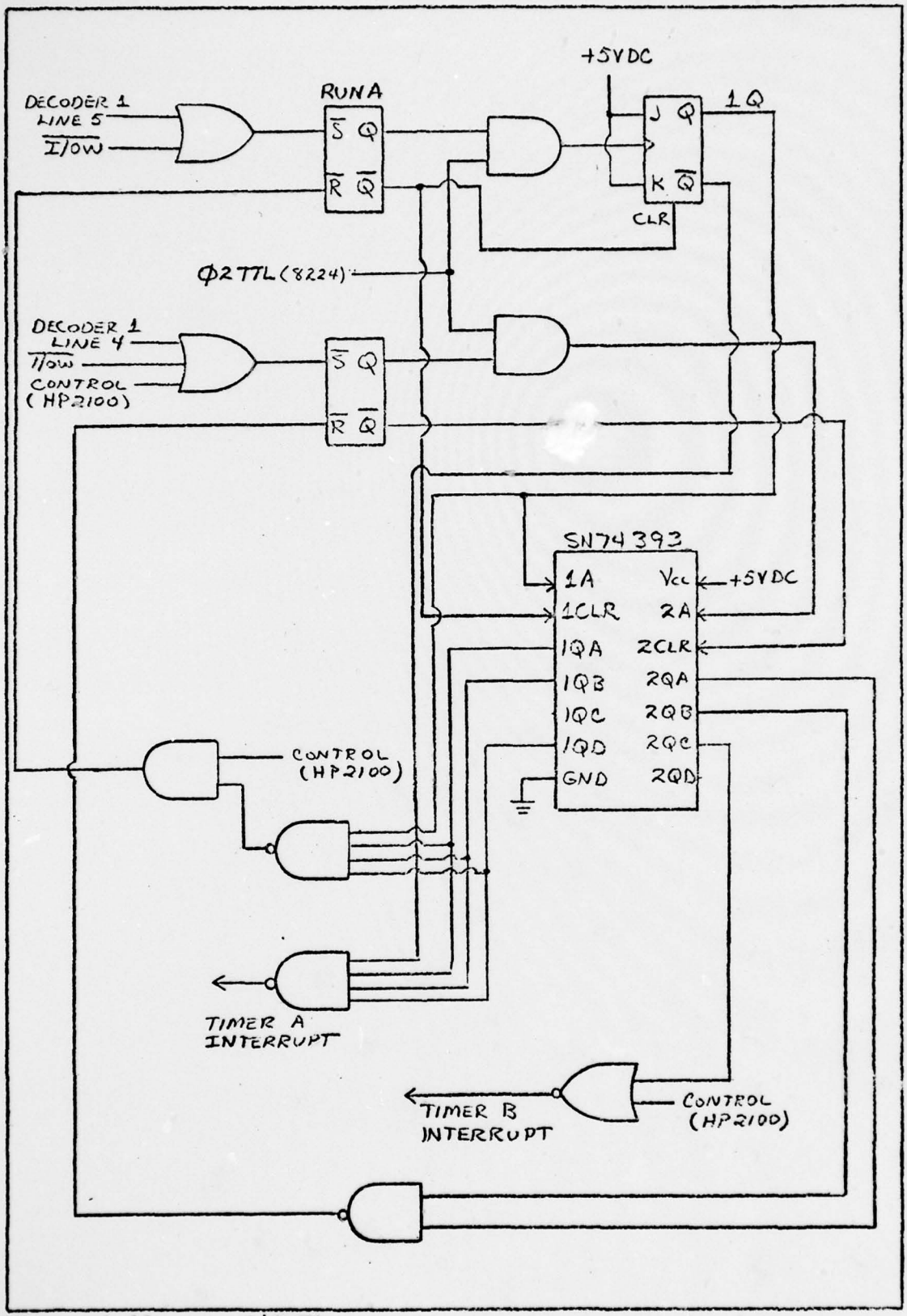


Figure 4-11 I/O Timers

two counters on one SN 74393, verifys that the HP 2100 acknowledges each word before the 8080A-1 begins to load the next word into the I/O ports. Timer B, the second counter on the SN 74393, keeps the 8080A-1 from attempting to set the FLAG when the CONTROL signal is low.

A RUNA flip-flop is used to control the operation of timer A. RUNA is set at the time a word is loaded into I/O port B (at the same time the FLAG is set); RUNA is reset when the timer runs out or when CONTROL goes low. Equations for RUNA are

$$\overline{S} = (\text{Decoder 1 Line 5}) + \overline{I/OW} \tag{3}$$

$$\overline{R} = \overline{1Q \cdot 1QA \cdot 1QB \cdot 1QD} \cdot (\text{CONTROL}) \tag{4}$$

When RUNA is set, the TTL clock signal from the 8224 is gated to a JK flip-flop which is used to extend the range of the binary counter in the SN 74393. The JK toggles at each clock pulse, and the Q output of the flip-flop drives the counter for timer A. The interrupt request line from timer A goes low after 21 clock pulses, and an interrupt is generated when the interrupt request line goes high at the end of the 22nd clock pulse.

$$\text{TIMER A} = \overline{1Q \cdot 1QA \cdot 1QB \cdot 1QC} \tag{5}$$

The RUNB flip-flop is used to control timer B which must run when CONTROL goes low properly but does not return high immediately. RUNB is set when a byte is loaded into I/O port A and CONTROL is low; RUNB is reset when CONTROL goes high or the timer runs out, whichever occurs first. For RUNB

$$\overline{S} = \overline{I/OW} + \overline{(\text{Decoder 1 Line 4})} + \text{CONTROL} \quad (6)$$

$$\overline{R} = \overline{QC + (\text{CONTROL})} \quad (7)$$

The TIMER B interrupt occurs at the end of the fourth clock pulse.

$$\text{TIMER B} = \overline{2QA \cdot 2QB} \quad (8)$$

Both timers and the JK flip-flop are cleared by the $\overline{Q}$ output of their respective run flip-flops.

Figure 4-12 provides a timing chart that shows the conditions which cause both I/O timers to generate interrupts. The chart is based on the following instruction sequence which is repeated for each data word:

```
MOV B, M    ; get one data byte
INX H       ; increment the index register
MOV A, M    ; get the next data byte
INX H       ; increment the index register
OUT IOPTA   ; write to I/O port A
MOV A, B    ; move high order byte to accumulator
OUT IOPTB   ; write to I/O port B
```

This sequence provides the longest period of time between the last byte of one 16 bit word (in I/O port B) and the first byte of the next word (in I/O port A), while still meeting the maximum data transmission rate. The longer the time period, the easier it is to synchronize the time digitization system and the HP 2100.

Following the timing chart provides a review of the functions of the I/O timers. RUNA (and the FLAG) is set when a byte is written into port B (by $\overline{I/OW}$). Timer A begins counting and generates an interrupt if CONTROL remains high indicating the HP 2100 has not received the new word. The interrupt stops the 8080A-1 before it

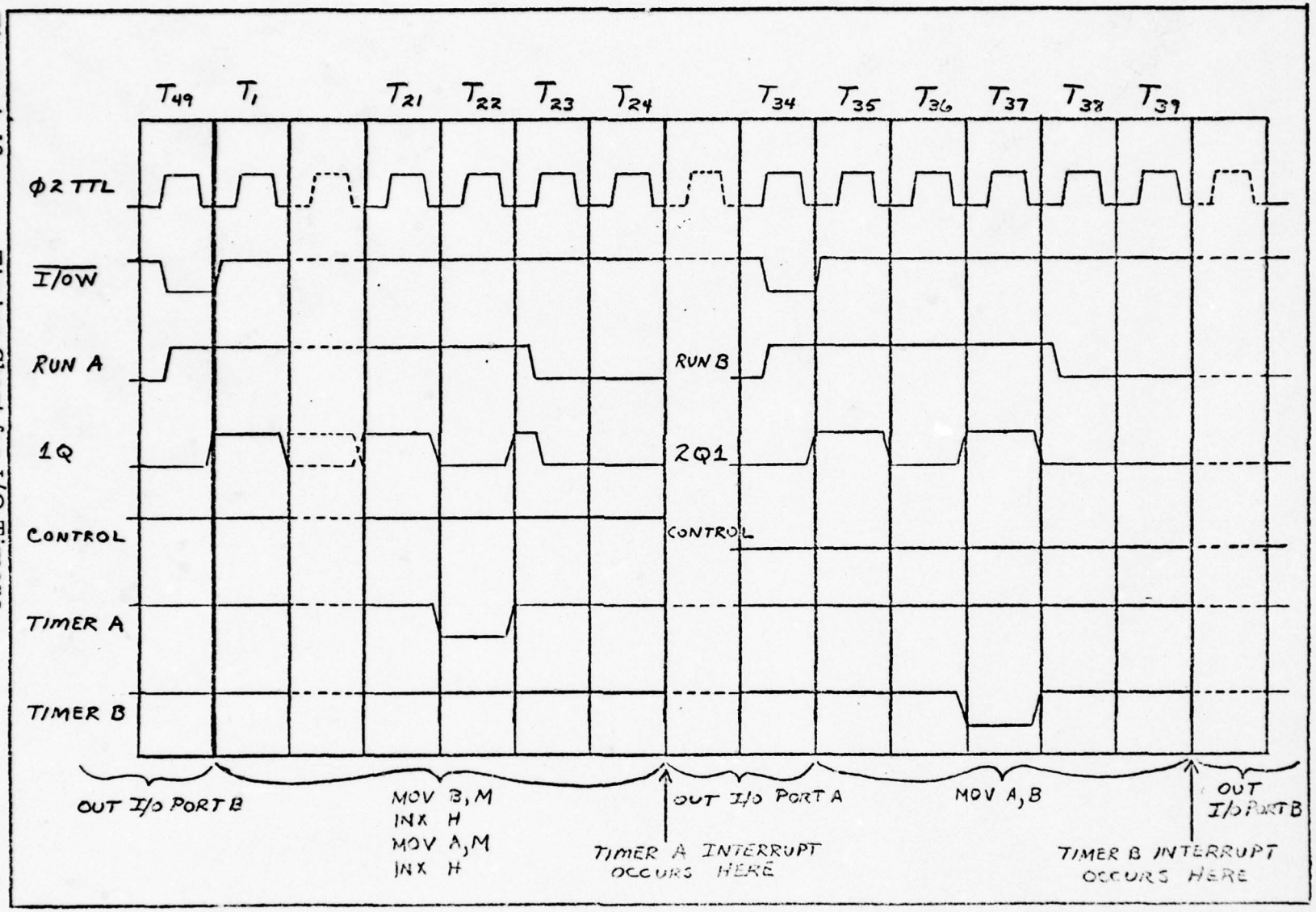


Figure 4-12 Timing Chart for I/O Timers

writes over the word still in the I/O ports. When CONTROL is low, it is safe to write into I/O port A ($\overline{\text{I/OW}}$ in T34), but the 8080A-1 must stop before writing into port B. This is because the FLAG can not be set while CONTROL is low, and without the FLAG, the minicomputer can not identify the next data word. Thus timer B begins when RUNB is set by a write to port A, and an interrupt is generated in time to stop the OUT IOPTB instruction if CONTROL stays low.

The purpose of this chapter was to select a processor to perform the functions of the Subsystem 1 design model, and to develop the circuitry for the processor. An eight bit microprocessor was determined to be the best type of processor. The Intel 8080A-1 was finally selected because it can meet the time digitization system's specified data transmission rate, and because it is most familiar to the designer. Support chips were added to the processor circuitry so the 8080A-1 can operate at maximum speed, and so it can interface properly with the minicomputer and with the Subsystem 2 hardware. In the next chapter, the Subsystem 2 design model is implemented.

V Hardware Selection and Circuit Layout (Subsystem 2)

This chapter completes the hardware design for the time digitization system. Hardware is selected to perform the functions described in Subsystem 2 of the SA design model, block diagrams are developed to show a high level view of Subsystem 2 organization, and detailed circuit specifications are prepared so the hardware can be built and tested.

The functions that are grouped in Subsystem 2 are those with timing restrictions which make special purpose hardware necessary. These functions require separate hardware devices, and the method of implementing any one function is usually independent of the method used for another. Thus, the design of Subsystem 2 differs from that of Subsystem 1. In Subsystem 1, one central piece of hardware, the 8080A-1 CPU, performs virtually all the functions, assisted by support chips which are specially designed to work with an 8080 microprocessor. Subsystem 1 is an LSI design while Subsystem 2, on the other hand, is a medium scale or MSI design. This means that hardware selection for Subsystem 2 is on a piece by piece basis, and that combinational logic as a design tool is more important.

Although more individual hardware devices must be selected for Subsystem 2, the SA design model provides more direct guidance here than for Subsystem 1. As noted in the concluding section of

Chapter III, the conversion of many of the Subsystem 2 SA nodes to hardware is a direct and obvious process. In other cases, the conversion can be done in several ways, and experience helps considerably in selecting the most efficient hardware implementation.

In selecting hardware, the overriding consideration is the need for speed, a result of the specified 40 MHz clock rate. A 40 MHz clock rate allows 25 nanoseconds (ns) for signals to propagate through combinational logic between clock pulses. Among TTL devices, only the high performance Schottky chips can meet the speed requirements, and virtually all of the Subsystem 2 hardware is of this type. Other types of hardware, ECL for example, are far more expensive, have fewer MSI functions available, are more difficult to use, and can be less reliable so they are not considered for this design. A consequence of the high speed required of the time digitization system is increased system cost: Schottky chips are more expensive than other TTL chips; the limited time available for signal propagation because of the fast clock makes more chips necessary; and power requirements are greater. However, if high performance is necessary to accurately evaluate existing or proposed inertial components, then the additional expense may be justified.

The remainder of this chapter is divided into two sections. In the first, Subsystem 2 Organization, hardware selection for functions in the SA design model is combined with the development of block diagrams. The block diagrams show electrical connections between the major

functional components of Subsystem 2 and the interfaces between Subsystem 1 and Subsystem 2. The second section, Subsystem 2 Circuit Layout, gives more details about the selected hardware implementations and shows the necessary TTL chips. Electrical specifications for the chips used in Subsystem 2 can be found in the TTL Data Book (Ref 14).

Subsystem 2 Organization

The overall organization of Subsystem 2 is shown by the diagram in Figure 5-1. The block diagram is derived directly from node A0, Figure 3-11, and the only new information is the relationship of Subsystem 1 I/O decoder outputs and data bus to the operation of Subsystem 2 hardware. In the following paragraphs, each of the five major functions of Subsystem 2 are developed in more detail.

Signal Conditioner. The Sled Test Automatic Reformating (STAR) system used by the Sled Test Section of the CIGTF employs a signal conditioner which can be applied to the time digitization system with only a few modifications. As the signal conditioner is not digital hardware, it is not in the author's area of specialty. It is recommended that personnel familiar with the STAR system develop a signal conditioner based on the specifications given in the SA design model.

Hardware Controller. Figure 5-2 shows the functions of the hardware controller with slightly more detail than node A2, Figure 3-13. Six 8-bit registers are used to store hardware control signals.

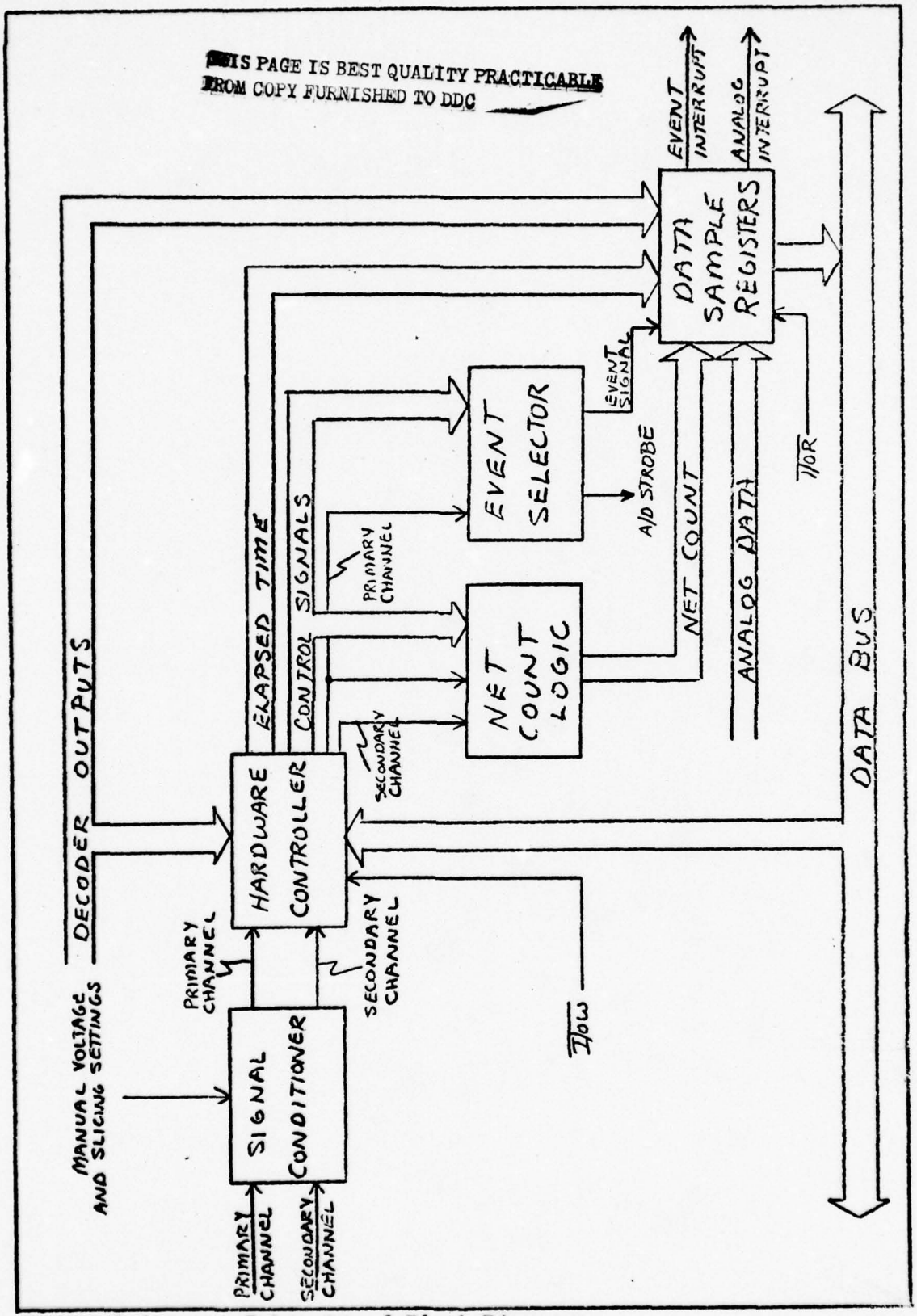


Figure 5-1 Subsystem 2 Block Diagram

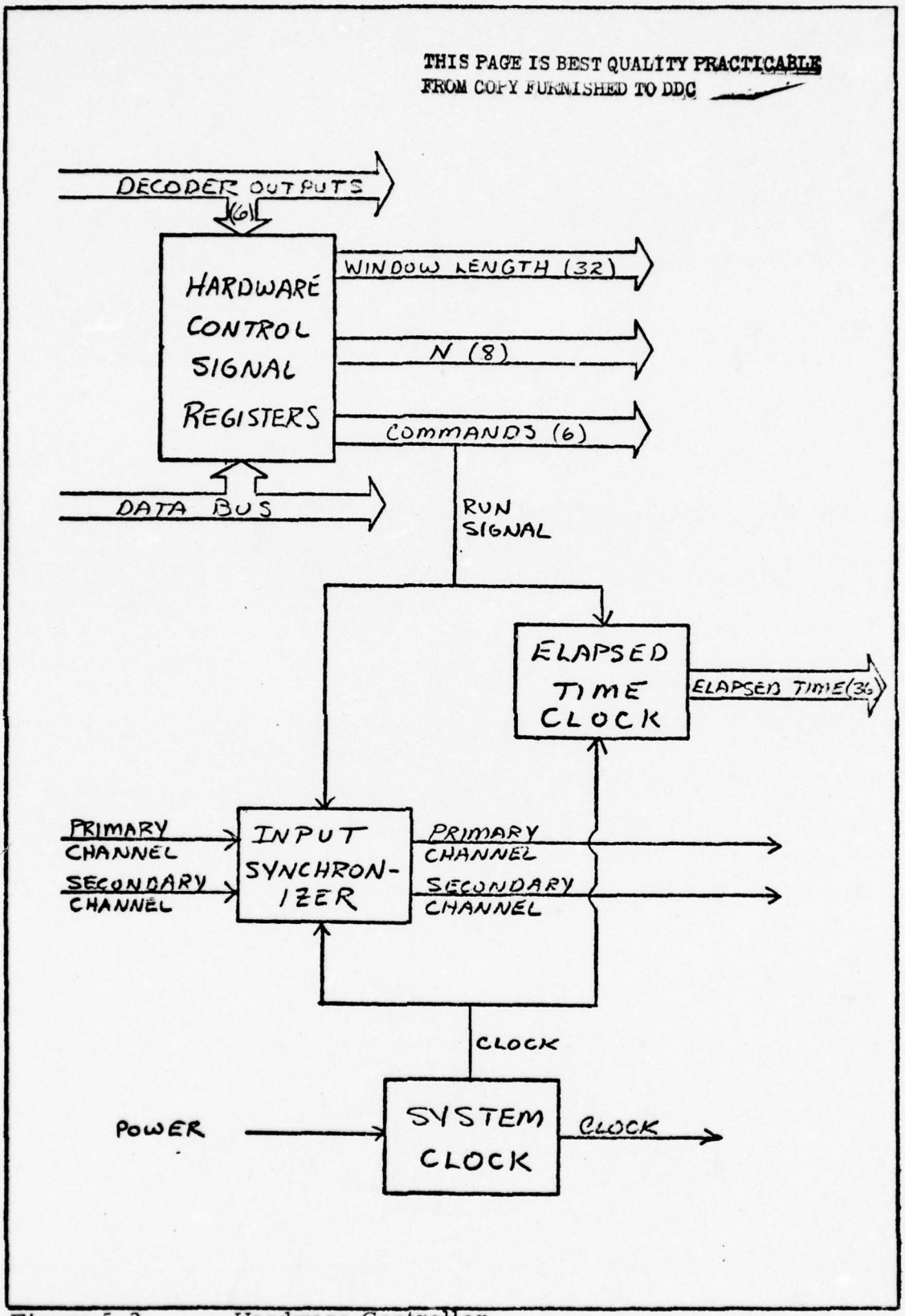


Figure 5-2 Hardware Controller

Four registers hold the nominal time window length when that option is used, and one register holds N for the pulse skip option. A command register provides individual signals for RUN, RESET, LEADING EDGE, TRAILING EDGE, PULSE SKIP, and TIME WINDOW. The elapsed time clock is a 36-bit counter which increments at each clock pulse when RUN is high. At 40 MHz, the elapsed time clock cycles about every 28 1/2 hours. In the input synchronizer, the primary and secondary channel digital signals from the signal conditioner are synchronized with the system clock. This minimizes the effects of noise on the input lines and allows the digital channel signals to be used with the clocked circuits in the remaining Subsystem 2 hardware.

The 40 MHz system clock operates any time power is applied to the system. To accommodate the remaining special purpose hardware, the clock must have a minimum pulse width of 10 ns and a maximum width of 15 ns. Figure 5-2 shows a clock output to the remainder of the system; however, from this point on the clock input to each module is assumed and not shown explicitly. The specifications for the system clock given in Chapter II demand very high performance. It is suggested that since the clock will be expensive, one copy should be purchased to drive all the copies of the time digitization system which are to be used together in one location. Furthermore, it is expected that the system clock will be designed by a contractor who specializes in digital clocks.

Net Count Logic. Using the functions established in node A3,

Figure 3-15, as a starting point, the net count circuits given here consist of combinational logic and an up/down counter. Combinational logic identified leading edges and trailing edges of pulses on the primary channel. The state of the secondary channel comes directly from the input synchronizer, and a flip-flop retains the state of the secondary channel at each pulse edge. Additional combinational logic is used to determine when the net count must be incremented or decremented. Net count overflow and underflow signals are also provided.

Another method of computing the net count is to use two counters, one for increments and one for decrements, and subtraction logic. This method might be slightly faster, but it is more complicated. Since the net count can change only once within a microsecond, speed is not so important in this function as it is for others.

Event Selector. The event selection hardware outlined in Figure 5-3 is derived from node A4, Figure 3-16. Combinational logic uses primary channel signals and command signals to identify the proper pulse edges. In the pulse skip option, pulse edge signals are used to decrement a counter which is preset with N, and when the counter reaches zero the next event causes an event signal. The window timer also uses down counters to accomplish the nominal time window option. At each event, the window timer is loaded with the window length and allowed to decrement with each clock pulse. The timer stops at zero and allows the next event to be signaled. It is assumed that the event signal also suffices as an A/D strobe. Although the largest N required

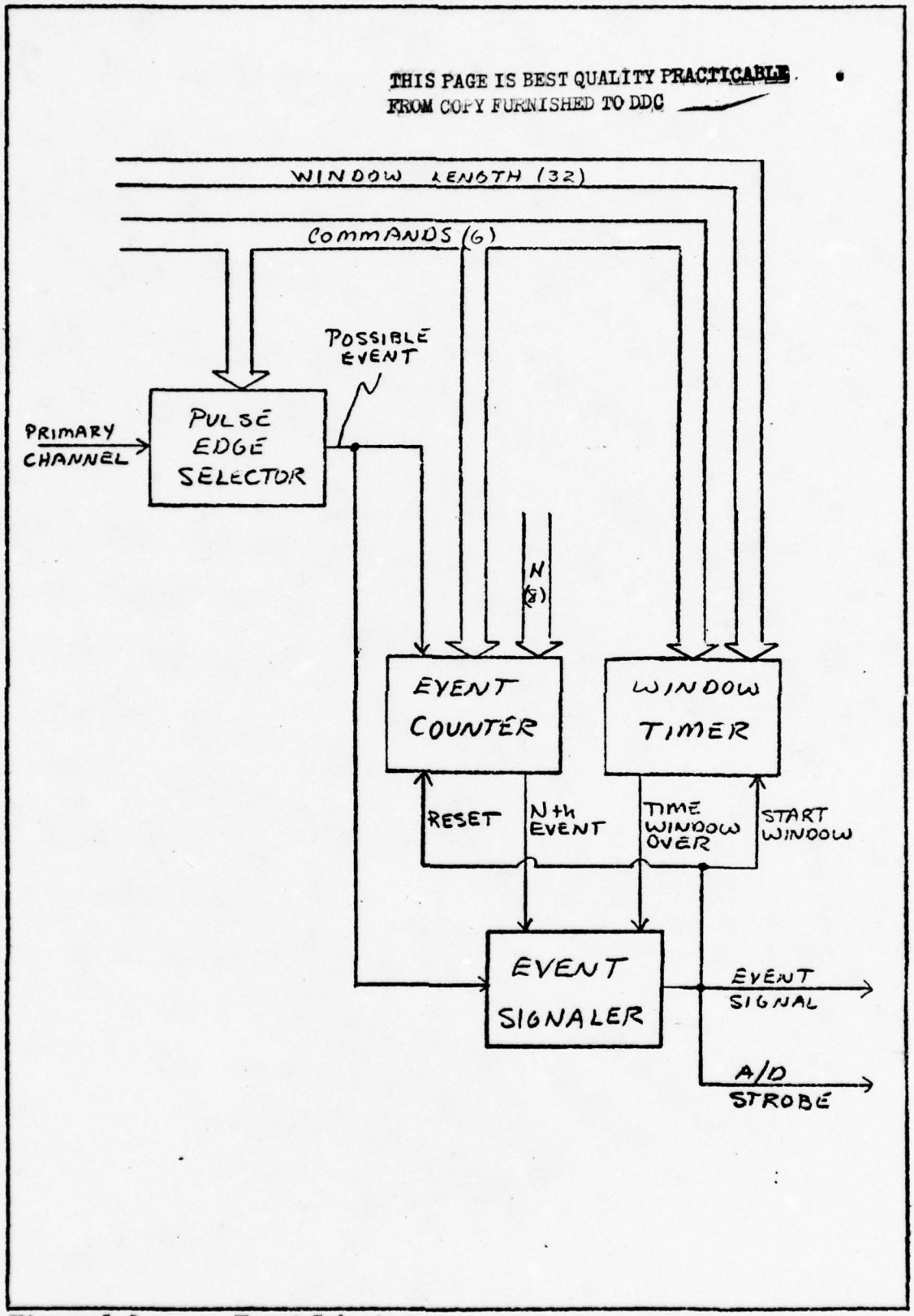


Figure 5-3 Event Selector

of the time digitization system is 127, the system actually allows N to be as large as 255. The nominal time window counters can operate with any value up to 107 seconds.

Preloading a counter and decrementing it to zero is more efficient than starting a counter at zero and using a comparator to determine when the selected N or time window has been reached. In fact, using comparators creates timing difficulties because of the 25 ns clock period, so down counters are used for both pulse skip and time window options.

Data Sample Registers. The registers used to store data for samples are given in Figure 5-4. This diagram adds a status register and interrupt request logic to the modules in node A5, Figure 3-17. All the registers are 8-bit, and in addition to the status register, five registers hold the event time, two hold the net count, and twelve hold analog data. Data is strobed into the registers by an event signal (delayed in the case of the net count) or by A/D end of conversion (EOC) signals. The same signals that strobe data into registers also cause interrupt requests through the interrupt logic.

This completes the hardware selection and organization, essentially a high level overview of how Subsystem 2 is implemented. The next section presents the details of the hardware and its operation.

Subsystem 2 Circuit Layout

In the following pages, each of the lowest level functions from the

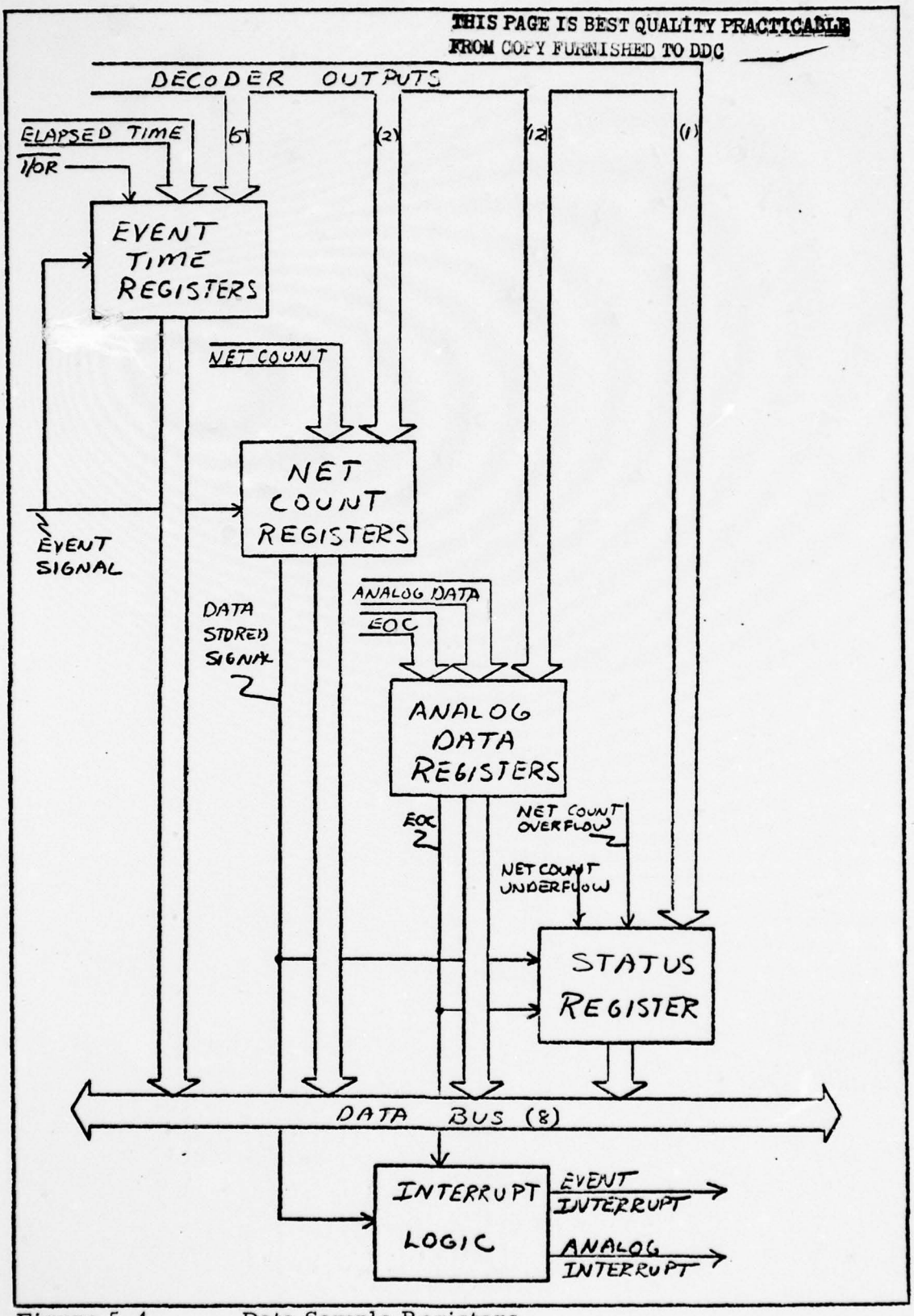


Figure 5-4 Data Sample Registers

Subsystem 2 block diagrams are presented separately. Where Texas Instruments integrated circuit (IC) numbers are given, it is important that those chips or their exact equivalents be used because the chips have been selected for their low signal propagation times. Where IC numbers are not provided, any class of TTL circuits may be used.

Hardware Control Signal Registers. The hardware control signal registers shown in Figure 5-5 are SN 74S373 octal D-type transparent latches (Ref 14:7.471-7.476). Data can be loaded into each latch from the data bus when the ENABLE G input for the register is high.

$$\text{ENABLE G} = \overline{(\text{Decoder 1 Line i})} \cdot \overline{\overline{(\text{I/OW})}}$$

$$= \overline{(\text{Decoder 1 Line i}) + \overline{\overline{(\text{I/OW})}}} \qquad (9)$$

All output control lines are tied low which allows the register outputs to be continuously available rather than in the high impedance state. For input to the registers, DB0 is connected to each 1D and so on through DB7 and 8D. Output signals are self explanitory. For N and the window value, low order bits are stored in the first flip-flop of each latch, and the window register A contains the low order byte of the time window.

Elapsed Time Clock. The elapsed time clock consists of nine SN 74S163 synchronous 4-bit binary counters (Ref 14:7.190-7.205). The elapsed time output, as shown in Figure 5-6, is labeled ET 1 through ET 36 with ET 1 being the lowest order bit. The counters operate any time RUN is high; for the counter enable P input in counter 1

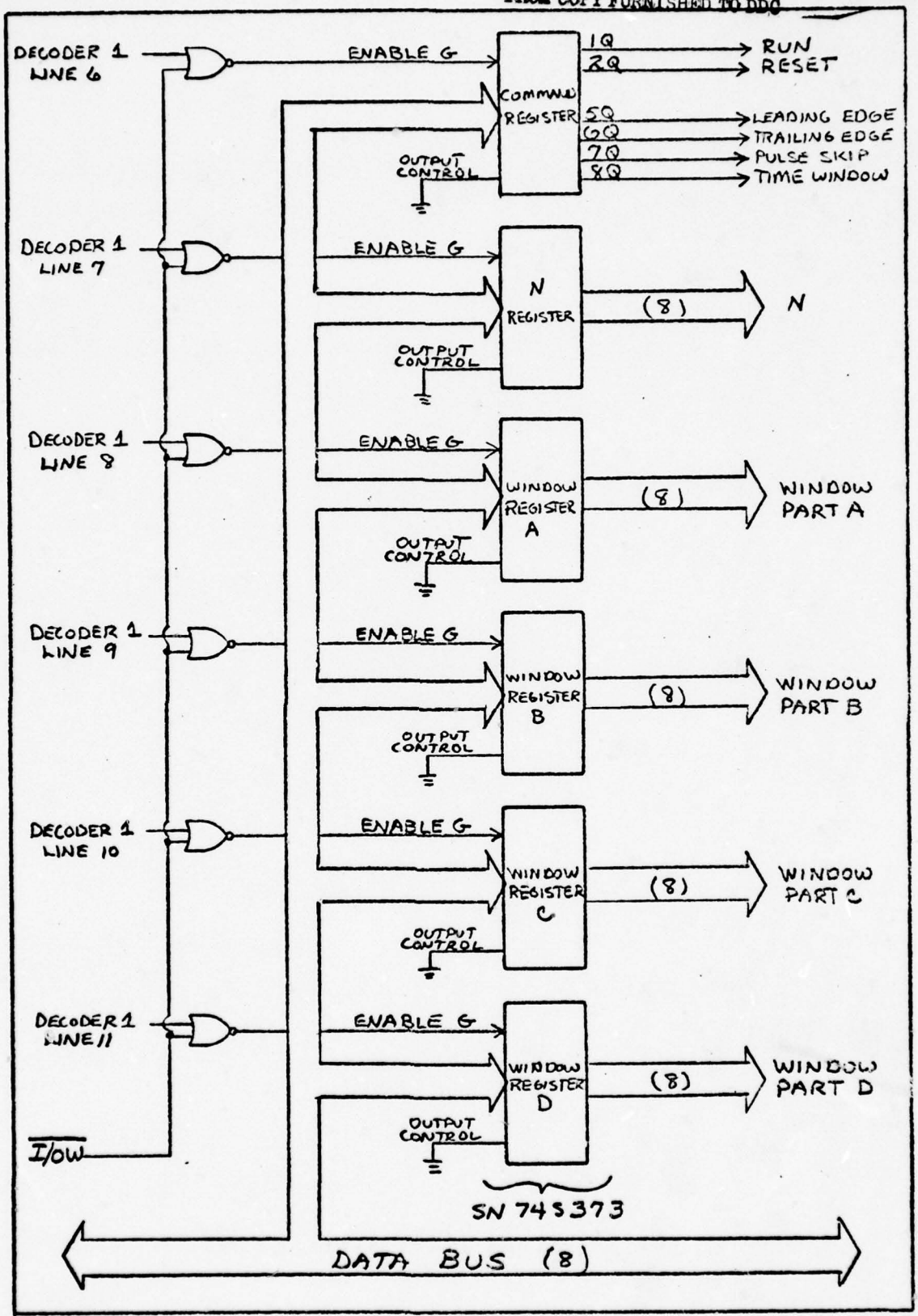


Figure 5-5 Hardware Control Registers

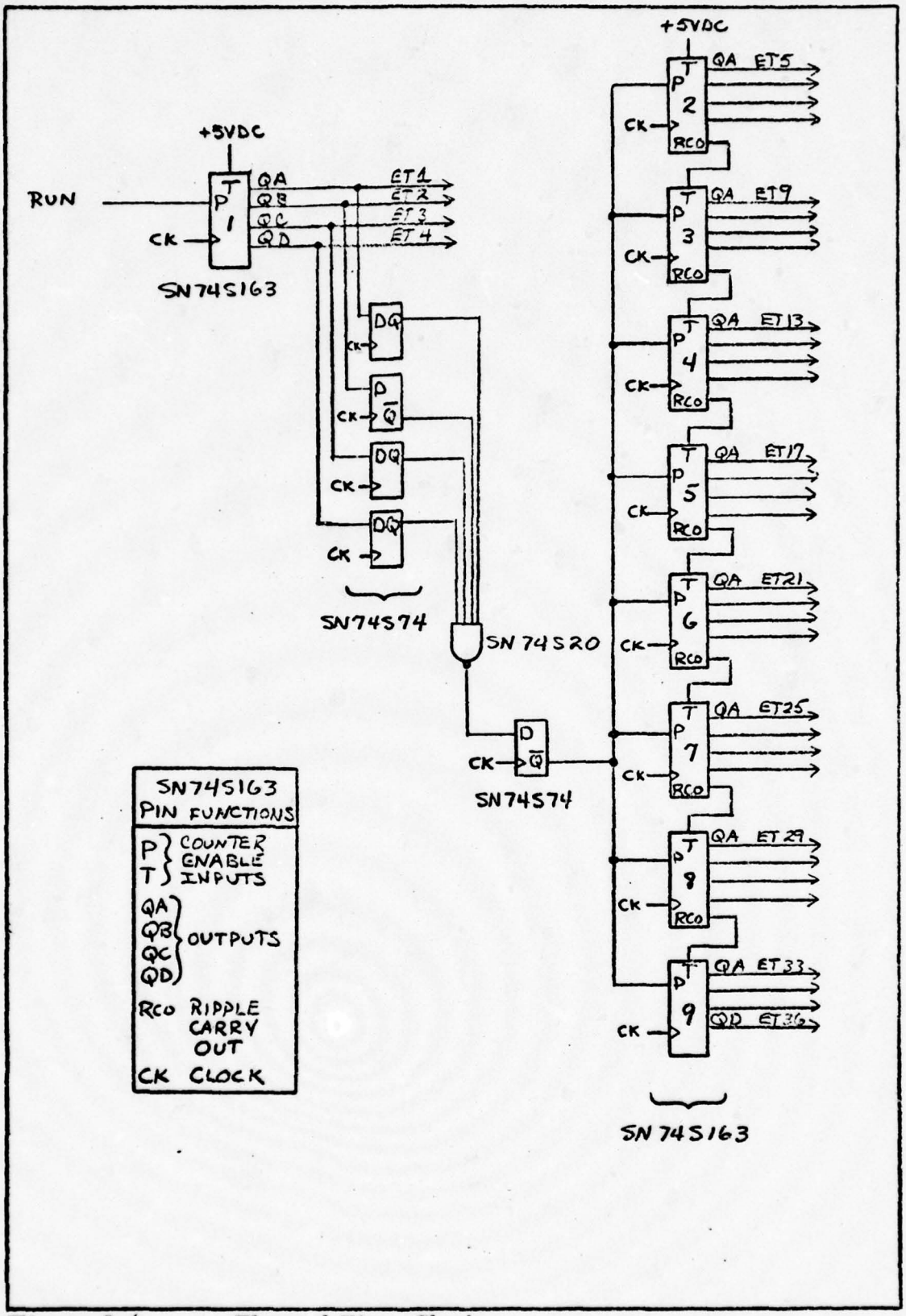


Figure 5-6 Elapsed Time Clock

$$P = RUN \tag{10}$$

Because the internal propagation time from the clock input to the ripple carry out (RCO) can be 25 ns, look-ahead-carry logic is necessary between the first and second counters. When the first counter reaches 1101 binary, the look-ahead-carry begins. After one clock period delay to accommodate counter propagation delays, an inverted carry signal is computed.

$$\overline{CARRY} = \overline{ET4(t-1)\cdot ET3(t-1)\cdot \overline{ET2(t-1)}\cdot ET1(t-1)} \tag{11}$$

After an additional delay to allow for set-up times for the P inputs, the carry signal is inverted once more through $\overline{Q}$ of a D flip-flop and used to activate the second through ninth counters.

$$\begin{aligned} P &= ET4(t-2)\cdot ET3(t-2)\cdot \overline{ET2(t-2)}\cdot ET1(t-2) \\ &= \overline{CARRY(t-1)} \end{aligned} \tag{12}$$

The T input to each counter must also be high to enable the counter. In addition, the T input enables the RCO signal as this is how the counters are cascaded. The T inputs for counters 1 and 2 are tied high and RCO's for counters 2 through 8 are connected to the T's of the succeeding counters. Ample time exists while the first counter completes a cycle for the RCO signal to propagate from the second to the ninth counters so no additional external look-ahead-carry logic is necessary.

Input Synchronization. Figure 5-7 shows the synchronization of the primary and secondary channel digital signals. When RUN is high, the conditioned digital signals drive D flip-flops, and the outputs with

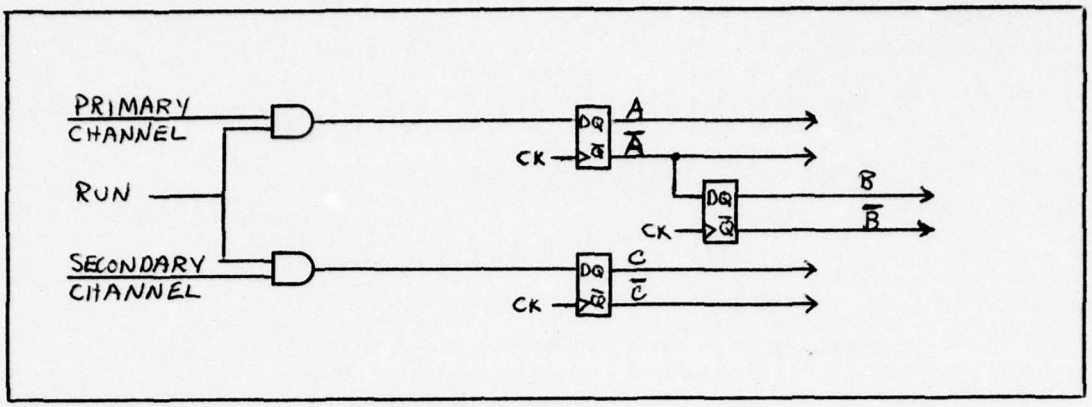


Figure 5-7 Input Synchronizer

reference to time are

$$A(t) = \text{Primary Channel State at } t-1 \tag{13}$$

$$C(t) = \text{Secondary Channel State at } t-1 \tag{14}$$

For the net count and event selection, the last state of the primary channel is retained:

$$B(t) = \text{Primary Channel State at } t-2 \tag{15}$$

The labels A, B, and C are retained through the remainder of the special purpose hardware: A is considered to be the current state of the primary channel, B is the state of A at the last clock pulse, and C is the current state of the secondary channel.

Net Count Logic Circuits. The net count is maintained by four SN 74S169 synchronous binary up/down counters (Ref 14:7.226-7.236). Figure 5-8 shows the net count logic circuits which are the most complex in the special purpose hardware. As a starting point for developing the circuits, equations (1) and (2) from Chapter II are repeated here. The symbol N which stands for a new or changed secondary channel state replaces the C used in the original equations to avoid confusion

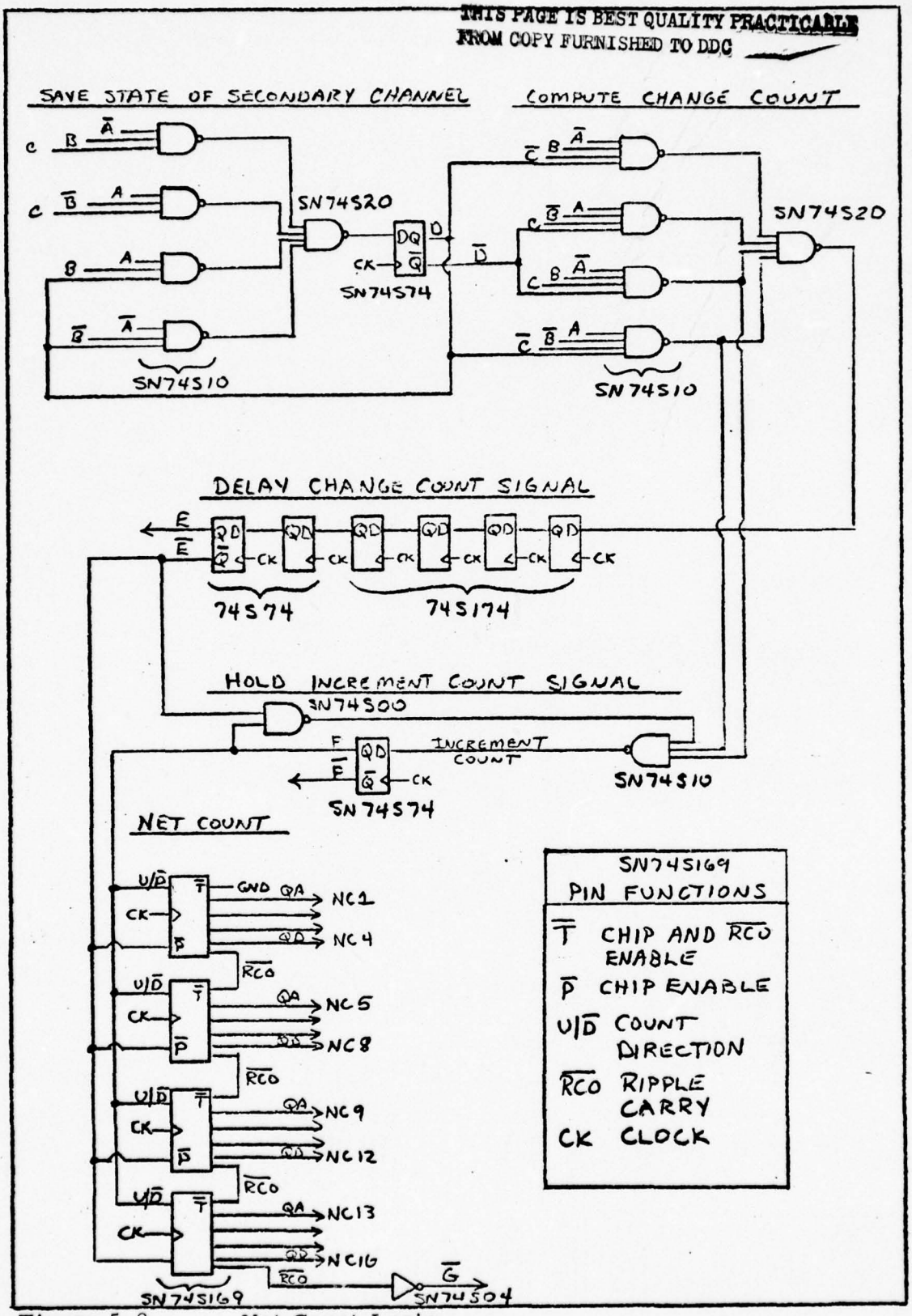


Figure 5-8 Net Count Logic

with the C used in Figure 5-7.

L = Leading Edge
T = Trailing Edge
S = Secondary Channel High
N = New Secondary Channel State

$$\text{Increment Count} = L\bar{S}N + TSN \tag{16}$$

$$\text{Decrement Count} = LSN + T\bar{S}N \tag{17}$$

What is necessary now is to develop the logic for identifying pulse edges, for detecting when a state change on the secondary channel has occurred, and for operating the counters.

A pulse edge on the primary channel can be detected when the current state of the primary channel (A) and the last state of the primary channel (B) are different. Thus

$$L = A\bar{B} \tag{18}$$

$$T = \bar{A}B \tag{19}$$

S or secondary channel high is simply C, the current state of the secondary channel

$$S = C \tag{20}$$

To determine if the state of the secondary channel has changed since the last pulse edge*, it is necessary to use a flip-flop to store the secondary channel state each time a pulse edge occurs. Let P be a pulse edge and D be the output of a D flip-flop used for storing the secondary channel state. Then

$$P = \bar{A}B + A\bar{B} \tag{21}$$

* Any odd number of changes, 1, 3, 5, . . . , are equivalent to one change of state; any even number of changes are equivalent to no state change.

$$D(t+1) = PC + \overline{P}D$$

$$= (\overline{A}B + A\overline{B})C + \overline{(\overline{A}B + A\overline{B})}D$$

$$= \overline{A}BC + A\overline{B}C + ABD + \overline{A}\cdot\overline{B}\cdot D \qquad (22)$$

In Figure 5-8, the AND-OR gates for D are replaced by two levels of NAND gates which are equivalent logically but operate faster. By comparing C and D, a new state on the secondary channel since the last pulse edge on the primary channel can be identified.

$$N = C\overline{D} + \overline{C}D \qquad (23)$$

Expressions for incrementing and decrementing the net count can now be given in terms of available signals by combining equations (18) through (23)

$$\text{Increment Count} = L\overline{S}N + TSN$$

$$= A\overline{B}\cdot\overline{C}(C\overline{D} + \overline{C}D) + \overline{A}B\cdot C(C\overline{D} + \overline{C}D)$$

$$= A\overline{B}\cdot\overline{C}D + \overline{A}BC\overline{D} \qquad (24)$$

$$\text{Decrement Count} = LSN + T\overline{S}N$$

$$= A\overline{B}\cdot C(C\overline{D} + \overline{C}D) + \overline{A}B\cdot\overline{C}(C\overline{D} + \overline{C}D)$$

$$= A\overline{B}C\overline{D} + \overline{A}B\overline{C}D$$

The SN 74S169 counters must be enabled for one clock pulse whenever the net count is to change. As a change in count is either an increment or a decrement,

$$\text{Change Count} = A\overline{B}\cdot\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}C\overline{D} + \overline{A}B\overline{C}D \qquad (25)$$

The change count signal in Figure 5-8 is computed using NAND gates in place of the AND and OR gates indicated in equation (25) again for speed.

The $U/\overline{D}$ input to the counters determines the direction of count, high for up and low for down. The default direction of the counters is selected to be down so the only directional signal necessary is a high on the $U/\overline{D}$ input when the count is to be incremented. Since the $U/\overline{D}$ input enables the counter's ripple carry output ($\overline{RCO}$), a high signal must be held on the $U/\overline{D}$ pin long enough for the carry to propagate through all the counters and the overflow/underflow logic before the count can be correctly changed. For this reason, also, the change count signal must be delayed. As the propagation time from $U/\overline{D}$ in the first counter to the overflow or underflow flip-flops can be up to 130 ns, the change count signal is delayed by six clock cycles (150 ns) using D flip-flops. Another D flip-flop, labeled F, is used to hold the increment count signal for the counters. The flip-flop is set by an increment count signal computed according to equation (24), and the high is held until the counters are changed.

$$F(t+1) = A\overline{B}\cdot\overline{C}D + \overline{A}BC\overline{D} + \overline{E}F \qquad (26)$$

where E is the delayed change count signal. Finally, the $\overline{E}$ signal is used to enable all four counters through their $\overline{P}$ inputs so the count can change at the next clock pulse.

The counters are cascaded using the $\overline{T}$ and $\overline{RCO}$ pins. The $\overline{T}$ for first counter is grounded which allows the first $\overline{RCO}$ to propagate whenever necessary. Each $\overline{RCO}$ is connected to the $\overline{T}$ of the following counter, and the last $\overline{RCO}$, labeled G, is used to compute overflow and underflow.

The output of the net count counters is correct after the eighth clock cycle following a pulse edge (200 ns). Since the minimum pulse width is one microsecond, errors in the count caused by pulse edges too close together are not anticipated. It should be added that the counter output is binary, and negative numbers are produced in two's complement form.

Figure 5-8 does not show provisions for clearing the SN 74S169 counters. Clearing is done by using the RESET signal from the command register to load zeroes into the counter data inputs. For all the counters, the data inputs are tied low and

$$\text{LOAD} = \overline{\text{RESET}} \tag{27}$$

Although not specified in the original requirements definition, net count overflow and underflow are recorded by the special purpose hardware to assist in data analysis. Figure 5-9 shows the logic for the overflow and underflow signals. The net count may be positive or negative so the first step in determining overflow and underflow is determining the state of the counters. An SR flip-flop is used to show that the counters are below zero (BZ). When operation begins, the net count is zero and the BZ flip-flop is reset. The counters go below zero at a change count signal E when the increment signal F is low and the carry G from counter four is also low. Thus for BZ

$$S = E \cdot \overline{F} \cdot \overline{G}$$

$$\overline{S} = \overline{E \cdot \overline{F} \cdot \overline{G}} \tag{28}$$

The BZ flip-flop must be reset when the counters become non-negative. For this

$$R = E \cdot F \cdot \overline{G}$$

$$\overline{R} = \overline{E \cdot F \cdot \overline{G}} \qquad (29)$$

Net count overflow occurs when the counters are above zero, the count is incremented, and the carry from counter four is low. For the SR flip-flop which shows overflow

$$\overline{S} = \overline{\overline{BZ} \cdot \overline{G} \cdot E \cdot F} \qquad (30)$$

Underflow occurs when the counters are below zero, the count is decremented, and the carry from counter four is low. For the underflow flip-flop,

$$\overline{S} = \overline{BZ \cdot \overline{G} \cdot E \cdot \overline{F}} \qquad (31)$$

Both the overflow and underflow flip-flops are reset by the RESET signal.

$$\overline{R} = \overline{RESET} \qquad (32)$$

Pulse Edge Selector. Equations (18) and (19) identify leading edges and trailing edges. Using the control signals LEADING EDGE and TRAILING EDGE from the command register, a possible event can be selected by

$$PE = A\overline{B} \cdot (\text{LEADING EDGE}) + \overline{A}B \cdot (\text{TRAILING EDGE}) \qquad (33)$$

Converting this equation to NAND gates gives the circuit of Figure 5-10. The D flip-flop is added to the circuit because of propagation delays through the two levels of NAND gates.

Event Counter. The event counter, Figure 5-11, uses two of the

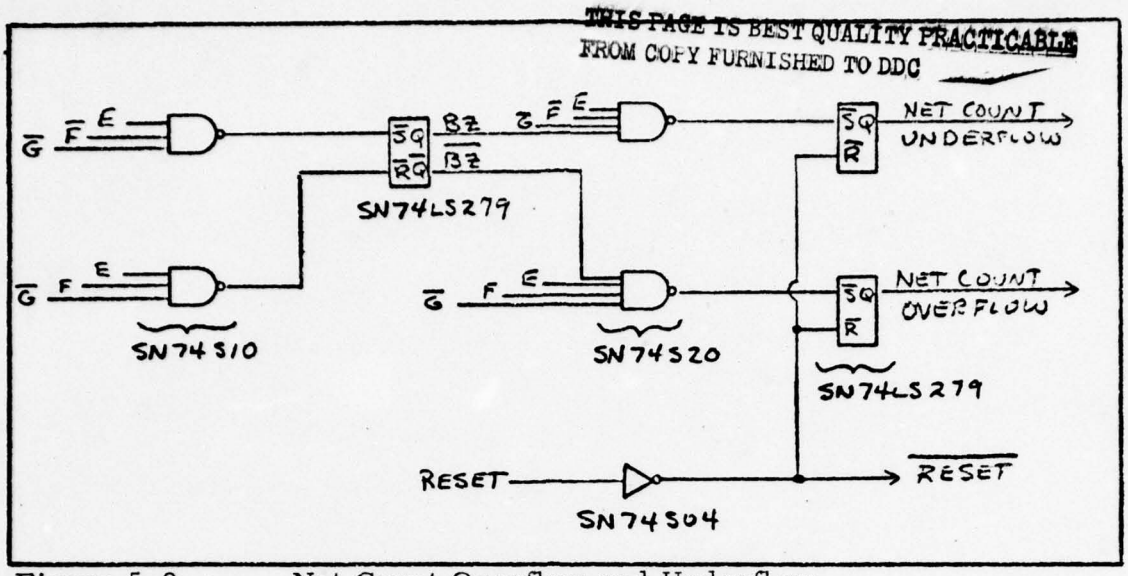


Figure 5-9 Net Count Overflow and Underflow

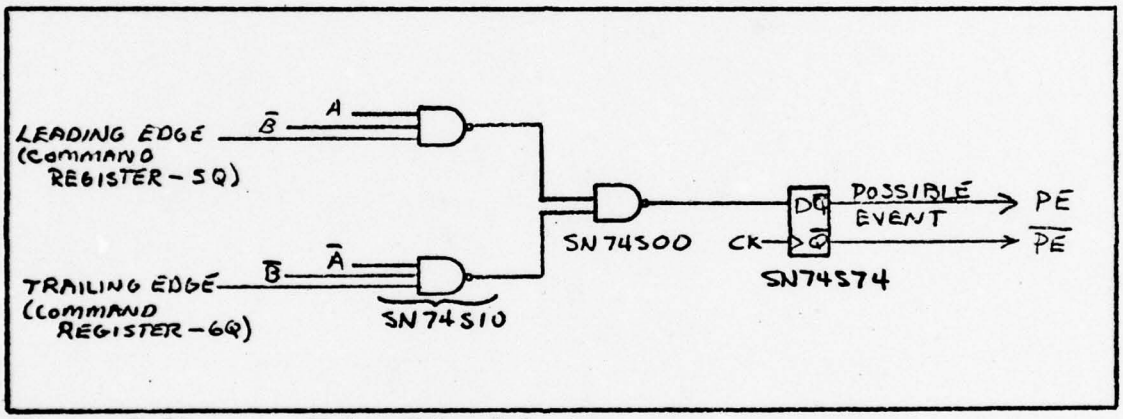


Figure 5-10 Pulse Edge Selector

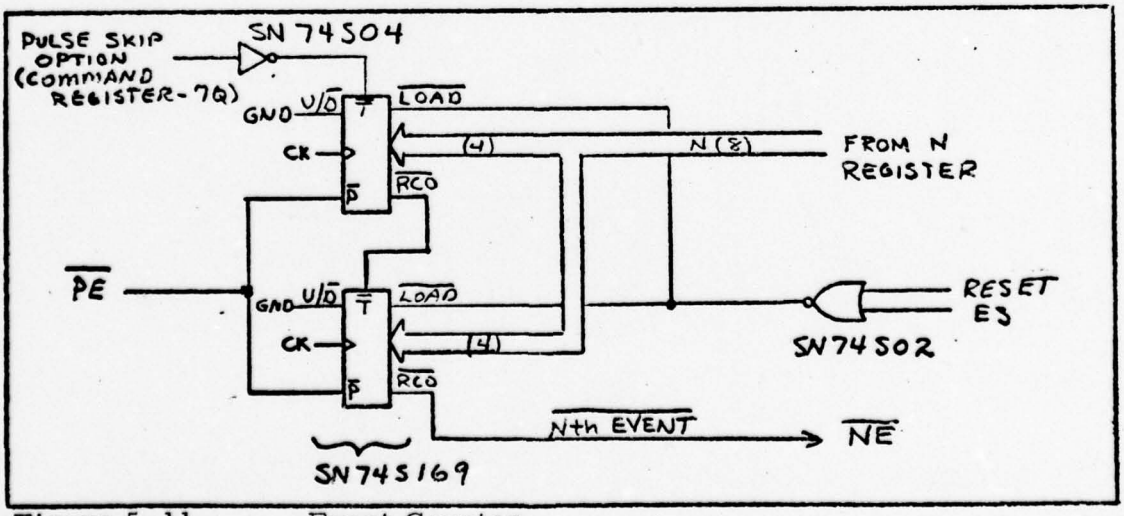


Figure 5-11 Event Counter

SN 74S169 counters. The counters are loaded with the current N value from the N register when the special purpose hardware is reset, or at an event signal (ES).

$$\text{LOAD} = \text{RESET} + \text{ES}$$

$$\overline{\text{LOAD}} = \overline{\text{RESET} + \text{ES}} \qquad (34)$$

The data inputs for the counters are given by

$$\begin{aligned} 1A &= 1Q \\ 1B &= 2Q \\ 1C &= 3Q \\ 1D &= 4Q \end{aligned}$$

$$\begin{aligned} 2A &= 5Q \\ 2B &= 6Q \\ 2C &= 7Q \\ 2D &= 8Q \end{aligned}$$

where the Q's are the output lines from the N register. Except during loading, the counters decrement at each possible event (PE); hence $U/\overline{D}$ is tied low and the enable inputs $\overline{P} = \overline{PE}$. The PULSE SKIP option signal from the command register is inverted and used for the $\overline{T}$ input to the first counter. The $\overline{T}$ partially enables the counter and allows the carryout signal ($\overline{RCO}$) to propagate to the second counter as necessary. When both counters are zero, the $\overline{RCO}$ for counter 2 goes low, and this is used to indicate that N events have occurred (NE).

Window Timer. The window timer in Figure 5-12 also uses SN 74S169 counters. As in the event counter, the window counters are loaded with the time window value in the window registers when the hardware is reset or when an event signal occurs.

$$\overline{\text{LOAD}} = \overline{\text{RESET} + \text{ES}} \qquad (35)$$

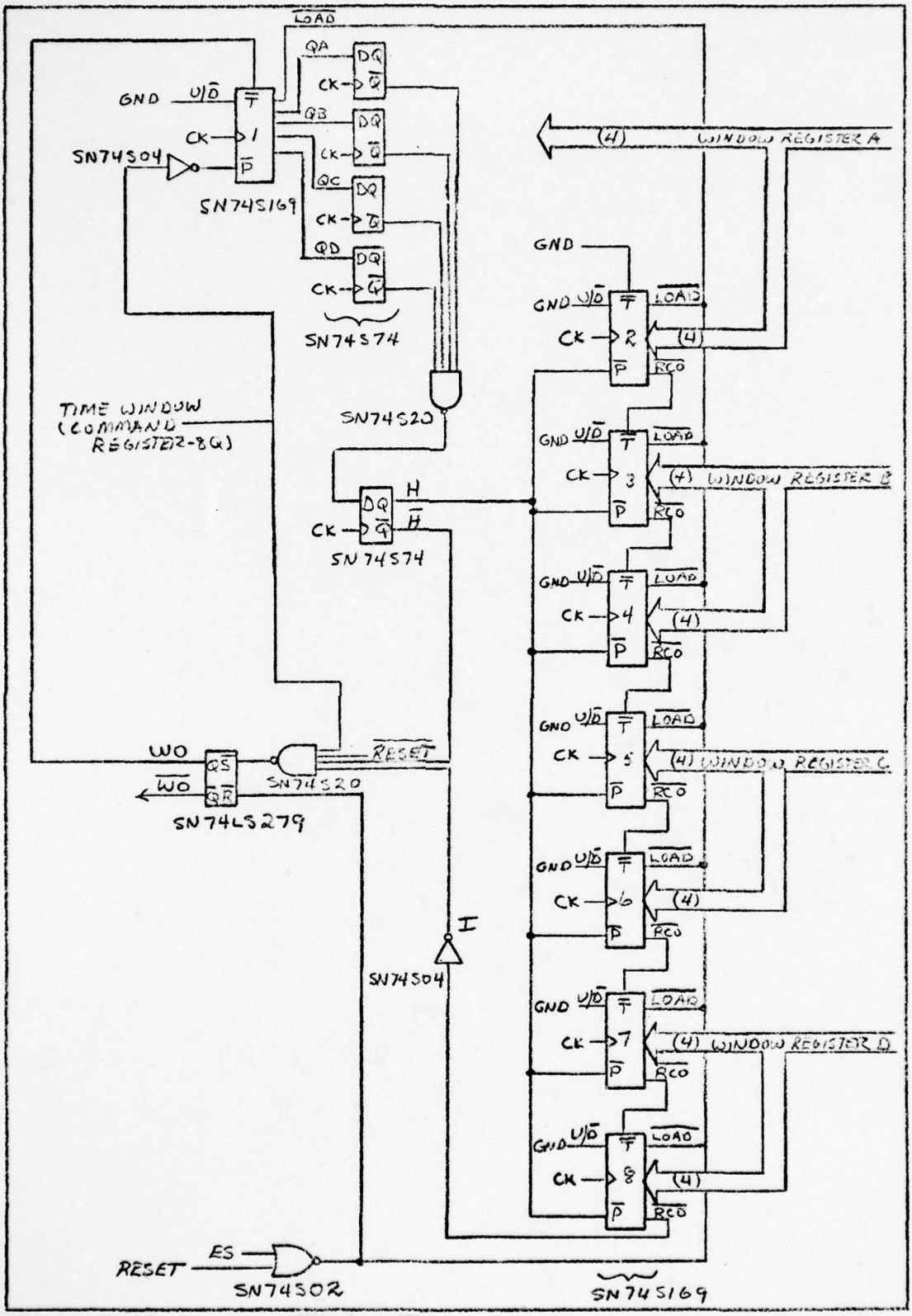


Figure 5-12 Window Timer

Window register A is used to load the first two counters with 1Q through 4Q connected to data inputs 1A through 1D of the first counter, and 5Q through 8Q connected to 2A through 2D of the second counter. The other window registers are used to load the remaining counters in a similar manner. To make the counters count down, $U/\overline{D}$ is tied to ground in all cases.

The counter enable pins $\overline{T}$ and $\overline{P}$ are used in several ways to get the proper count at the proper time. The $\overline{P}$ of the first counter is enabled by an inverted TIME WINDOW signal from the command register. The $\overline{P}$ input for the first counter is activated by a low signal from the Q output of a WINDOW OVER (WO) flip-flop. At each clock pulse, the value in the first counter is decremented by one, and when the count reaches zero, a carry-out signal (H) is generated. With the outputs of the counter labeled QA, QB, QC, and QD, then

$$H = \overline{\overline{QA}\cdot\overline{QB}\cdot\overline{QC}\cdot\overline{QD}} \qquad (36)$$

A low signal on H enables the $\overline{P}$ for all but the first counter. Two delays (D flip-flops) are necessary because of propagation times through the first counter and the NAND gate and the set-up time for $\overline{P}$. The delays cause the WINDOW OVER signal to be two clock pulses late so the processor must subtract 2 from the time window value when it is loaded in the time window registers. The $\overline{T}$ for the second counter is tied low, and the remaining $\overline{T}$ and the $\overline{RCO}$ pins are connected in the normal manner for cascading counters.

When the counters reach zero, the WO flip-flop is set. Provided

RESET is low and TIME WINDOW is high, the inverted carry-out signal (I) from the last counter and the carry-out ($\overline{H}$) from the first counter will set the WO flip-flop.

$$\overline{S} = \overline{I \cdot \overline{H} \cdot (\overline{RESET}) \cdot (TIME\ WINDOW)} \tag{37}$$

The WO flip-flop is reset at an event signal, or when the hardware is reset.

$$\overline{R} = \overline{ES + RESET} \tag{38}$$

Notice this is identical to the equation for $\overline{LOAD}$ (35).

Event Signaler. When N events have occurred (NE), or the time window is over (WO), then the next possible event (PE) creates an event signal (ES).

$$\begin{aligned} ES &= PE(NE + WO) \\ &= \overline{\overline{PE} + \overline{NE} \cdot \overline{WO}} \end{aligned} \tag{39}$$

This is shown in Figure 5-13. A D flip-flop is used to hold the ES signal for one clock cycle because of propagation time limitations.

Event Time Registers. When an event signal occurs, the event time must be stored in the event time registers, Figure 5-14. The SN 74S374 octal edge-triggered flip-flops (Ref 14:7.473-7.477) are the only eight bit registers with 3-state outputs which have data set-up times compatible with the 40 MHz clock. Unfortunately, these chips have only one enable input (CK) which creates timing problems. From a system clock pulse, the elapsed time clock counters can take 18 ns to stabilize. Another 5 ns are necessary for input set-up time in the SN 74S374. Thus the CK input must not be strobed until at least 23 ns

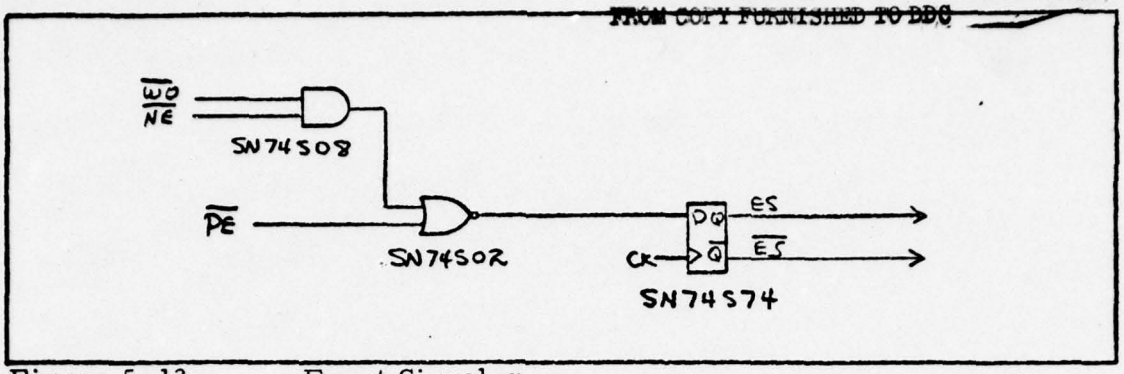


Figure 5-13 Event Signaler

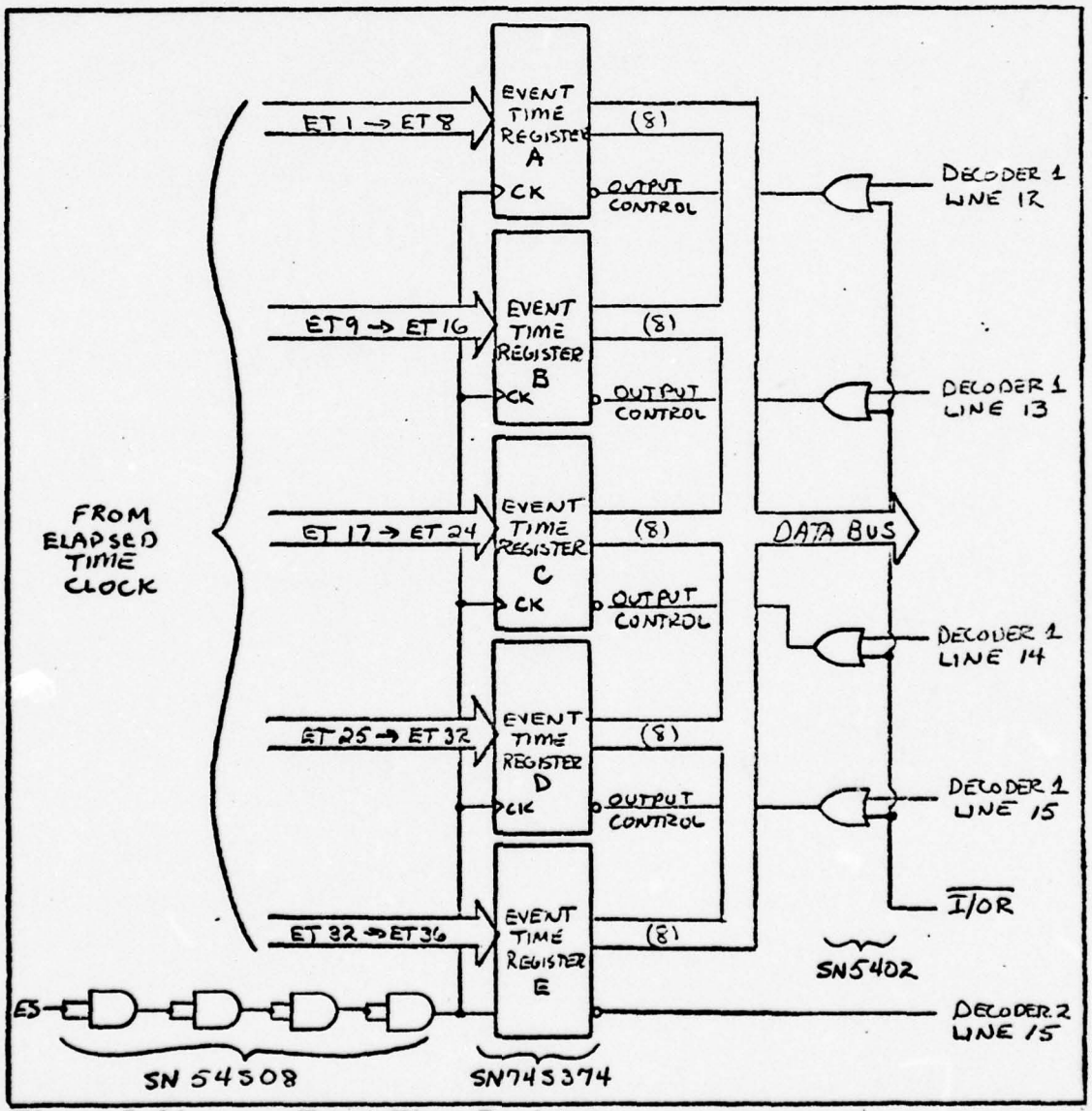


Figure 5-14 Event Time Registers

have elapsed following the leading edge of the clock pulse in which an event signal (ES) is generated. The SN 74S74 D flip-flop used for the event signal has a typical signal propagation delay time of 6 ns; the SN 74S08 AND gates have typical propagation delay times of 4.75 ns. The circuit as diagrammed has an approximate 25 ns delay from the system clock pulse to the strobe (CK) which stores the event time. When the system is built, this delay must be checked carefully to keep it within the range of 23 to 25 ns. After 25 ns, signals from the elapsed time clock may change because of the next system clock pulse.

The output control (OC) inputs for the event time registers are activated by the appropriate I/O decoder lines. The lines which come from decoder 1 are gated with the $\overline{I/OR}$ signal to avoid connecting a register with the data bus at the wrong time.

$$OC = \overline{I/OR} + (\text{Decoder 1, Line i}) \qquad (40)$$

Since decoder 2 is activated by $\overline{I/OR}$, the line from that decoder does not need to be gated.

Analog Data Registers. Twelve SN 74S374 registers are used to store the 96 bits from the 8 A/D converters (Figure 5-15). Several of the registers are split between A/D converters. For example, register B holds the high order four bits from the first A/D converter and the low order four bits from the second A/D converter. Because the A/D converters are not clocked, no timing problem such as that with the event time registers exists. The end of conversion (EOC) signals for the first four A/D converters are connected to an AND gate

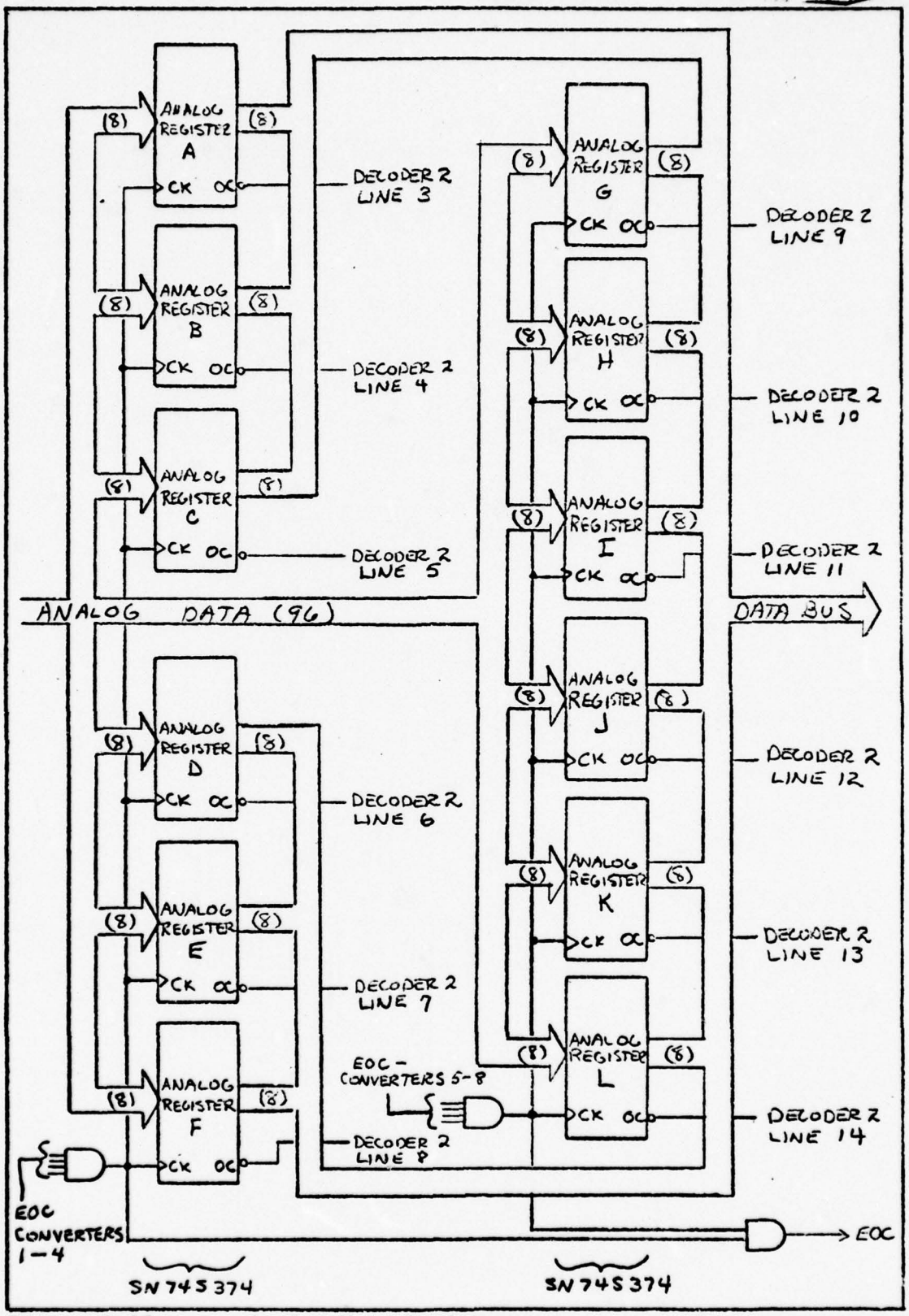


Figure 5-15 Analog Data Registers

to produce one EOC signal to strobe data into the first six analog data registers. The same process is used for the second four A/D converters, and a final system EOC signal is generated to show that analog data acquisition is complete. The assumption here is that all EOC signals from the A/D converters are high except during an actual conversion. If less than eight A/D converters are used, the unused EOC lines should be tied high. Appropriate lines from decoder 2 are used to select the analog data registers when data is read into the 8080 CPU.

Net Count Registers. Two more SN 74S374 registers are used to store the net count (Figure 5-16). Since the net count remains stable between pulse edges, no special delay elements like those used for the elapsed time registers are required. However, after a pulse edge appears at the input synchronization flip-flops, eight clock periods must elapse before the net count is computed. An event signal appears after only three clock cycles so an additional five clock period delays are necessary before the net count can be strobed into the net count registers. This explains the five D flip-flops used in Figure 5-16. At the time data is strobed into the net count registers, a data stored ($\overline{DS}$) signal is produced for the interrupt request logic and the status register. The net count registers put their contents on the data bus when selected by lines 1 and 2 from decoder 2.

Status Register. The status register, Figure 5-17, is another SN 74S374. Net count overflow and underflow signals feed the register

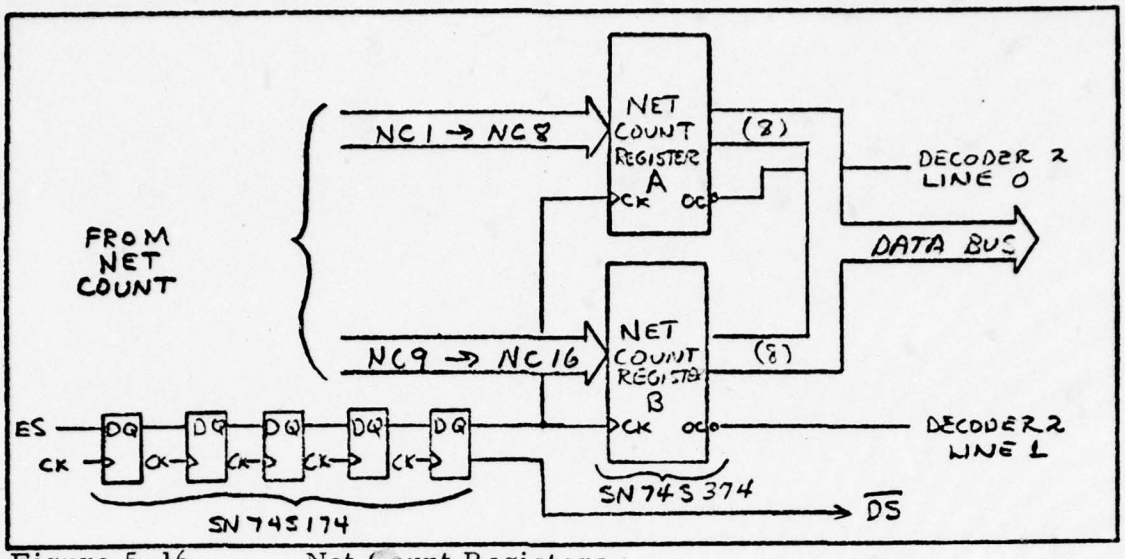


Figure 5-16 Net Count Registers

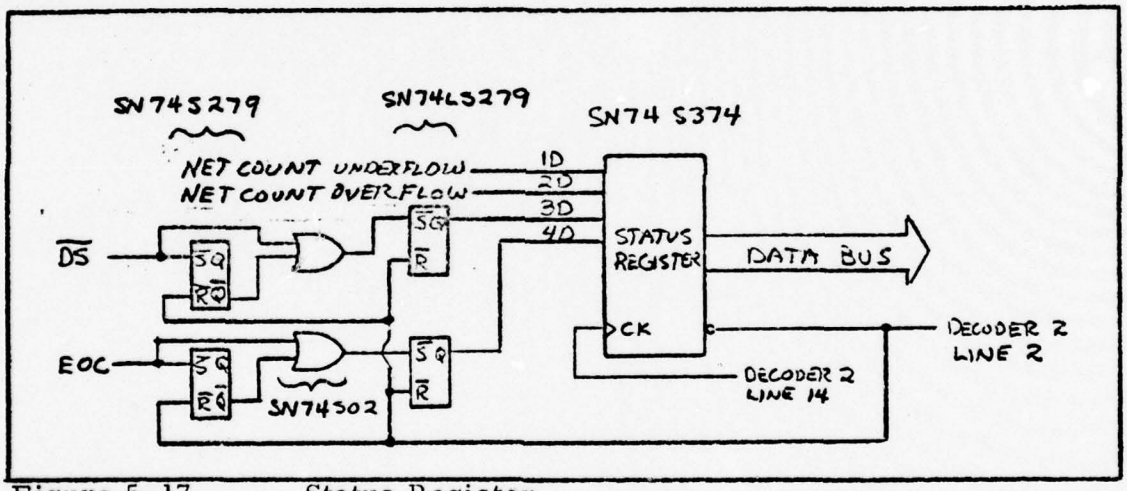


Figure 5-17 Status Register

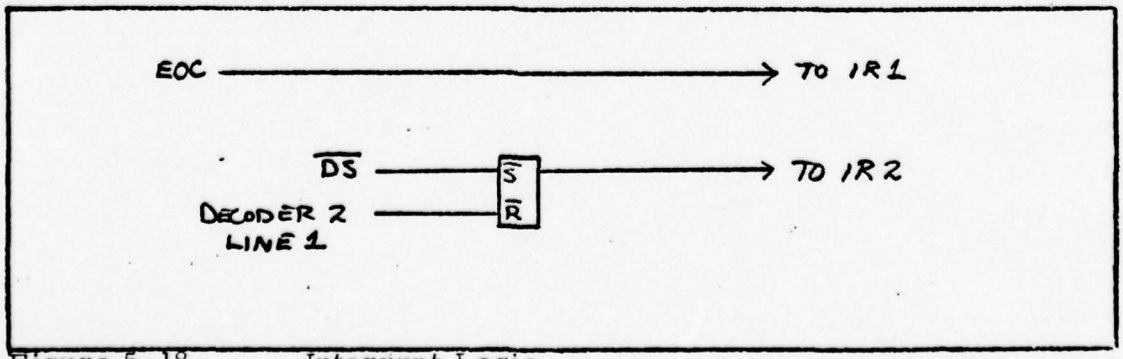


Figure 5-18 Interrupt Logic

directly. One pair of SR flip-flops is used to detect lost event data (3D), and another pair detects lost analog data (4D). For lost event data, a $\overline{DS}$ signal sets the first flip-flop. A second $\overline{DS}$ before the status register is read will set the second flip-flop to indicate that the data in the net count and elapsed time registers has been overwritten. The same process is used to show lost analog data with a low EOC signal replacing $\overline{DS}$. Status signals are strobed into the status register at the same time data is read from analog data register L. When the status register is read, all the SR flip-flops are reset. Note that the status register is to be read after the analog data from an event is read.

Interrupt Logic. Figure 5-18 shows the simple interrupt request logic. The EOC signal from the analog data register hardware should be adequate to signal an *ANALOG interrupt request.* The EVENT interrupt request is provided by an SR flip-flop which is set when the net count is stored ($\overline{DS}$) and reset when data is read from the last net count register.

Signal Propagation Delays. Signal propagation delays through the components of the special purpose hardware are an important consideration in the hardware design. Another factor which will become important when the circuits are constructed is the propagation time through the wires between components. Due to the large number of chips which are necessary, it is quite likely that more than one circuit board will be needed. Careful consideration must be given to which functions are placed on separate boards.

The purpose of this chapter was to select hardware for Subsystem 2 of the design model, to construct block diagrams showing how the hardware is related, and to develop the detailed circuit layout. The activities in the Subsystem 2 model lend themselves to simple MSI implementations; the basic hardware components are counters and registers. Because the Subsystem 2 design model reflected real hardware capabilities as well as system requirements, very little effort was necessary to convert the model to block diagrams. The circuit layout was complicated to a certain extent by the 40 MHz clock rate. The high frequency dictated the use of TTL Schottky chips and increased the chip count. It should be emphasized, however, that the performance specifications for data acquisition and time resolution can be met.

In the next chapter, the discussion shifts from hardware to software. A software structure is developed which can operate efficiently with the 8080A-1 and the special purpose hardware developed in this chapter.

VI Software Structure and Coding

A software structure is the form of organization imposed on all the segments of code which go together to make up a complete program. It often happens that code is written first, and later, perhaps, it is organized into interrelated modules. A more efficient approach is to develop a structure first, and then write the code. When the structure is developed first, it can be given some desirable qualities: the individual modules can be made relatively independent so they can be tested, corrected, or changed with little effect on other modules. The discipline imposed by a top down structure also helps assure that the program does all the functions that are necessary. Once a good structure is designed, the coding becomes almost mechanical.

Although it is easy to get software specialists to agree on the need for a software structure, no system for designing a structure seems to have universal acceptance. Structured Analysis is frequently used to develop software designs. Other methods include Structured Design, Jackson's Method, and an approach which treats programs as if they were finite state automata (Ref 15:1). The strength of SA in developing the time digitization system requirements definition and system design is its ability to show concurrent activities. Its weakness for developing the system software design is its inability to show activities occurring in a random order, but random activities are likely in the time

digitization system because events and minicomputer interrupts are independent. Structured Design has a similar weakness; its structure charts simply can not coordinate asynchronous interrupts. Michael Jackson's Method develops a program structure around data structures, and it is difficult to see how this could apply to the time digitization system. On the other hand, the finite state automata approach seems quite natural for this type of interrupt driven system.

The time digitization system collects or transmits data following random interrupts. Each interrupt puts the system into a new condition possibly requiring a different response to subsequent interrupts. Thus state diagrams and state tables present a good picture of what the system must do at various times.

The state diagrams and state tables used in this design are comparable to those used for Mealy type finite state machines. Each input or interrupt in this case causes a state transition, and during the transition an output is produced. The output here is some processing by the time digitization system, and the processes associated with each state transition are the modules of this design. The structure within each module is generally quite simple and can be described with flow charts.

Code can be derived directly from the flow charts without much difficulty. It is expected that the final coding for the system will be done in assembly language. Although slower to develop than higher order language programs, assembly language can be more efficient in

execution which saves processing time and ROM, both critical in this design.

This chapter is much less detailed than the hardware selection and circuit layout chapters. The reason is that several major design decisions are yet to be made: the interface between the HP 2100 and the time digitization system; the number of data records to be stored and transmitted at one time; and even the need for some of the critical timing specifications. All of the decisions affect the software structure. A change in the interface method can change the number and purpose of several interrupts. Additional RAM would change the frequency of interrupts, and a change in timing specifications might change some of the hardware, invalidating much of the software structure. At this point it is not efficient to establish much more than the general software structure, so very little code is included here.

This chapter contains a software structure for the time digitization system as it was developed in the hardware selection and circuit layout chapters. The state diagrams and state tables are developed first. Then the formats for I/O between the minicomputer and the time digitization system are defined to help clarify the flow charts which are given in the final section.

Software Structure

The operation of the time digitization system can be divided into two phases, and the software structure is divided to reflect this. The

first phase is the system start-up; the hardware is cleared, commands are received from the minicomputer, and the special purpose hardware is set up for operation in the proper mode. The second phase, which is much more complicated, is data collection using the selected mode of operation; data is acquired, placed in the proper format, and transmitted to the minicomputer.

In the second period, four modes of operation are possible. These are defined as follows:

MODE A	Pulse skip option with N specified
MODE B	Pulse skip option with N unspecified
MODE C	Nominal time window option with the time window specified
MODE D	Nominal time window option with the time window unspecified

System Start-up. A state diagram for the start-up period is presented in Figure 6-1. Except for POWER ON, each state is labeled with a letter as well as a name to assist the conversion to a state table. Each state is a waiting state, and transitions are caused by interrupts. The labels on the transitions give the interrupt causing the transitions first, and then the process which is accomplished during the transition. A manual RESET signal causes the transition from POWER ON to state A where the system waits for the first interrupt from the HP 2100.

The first interrupt is CONTROL which comes after a word is placed in the I/O ports. The word must be decoded to determine

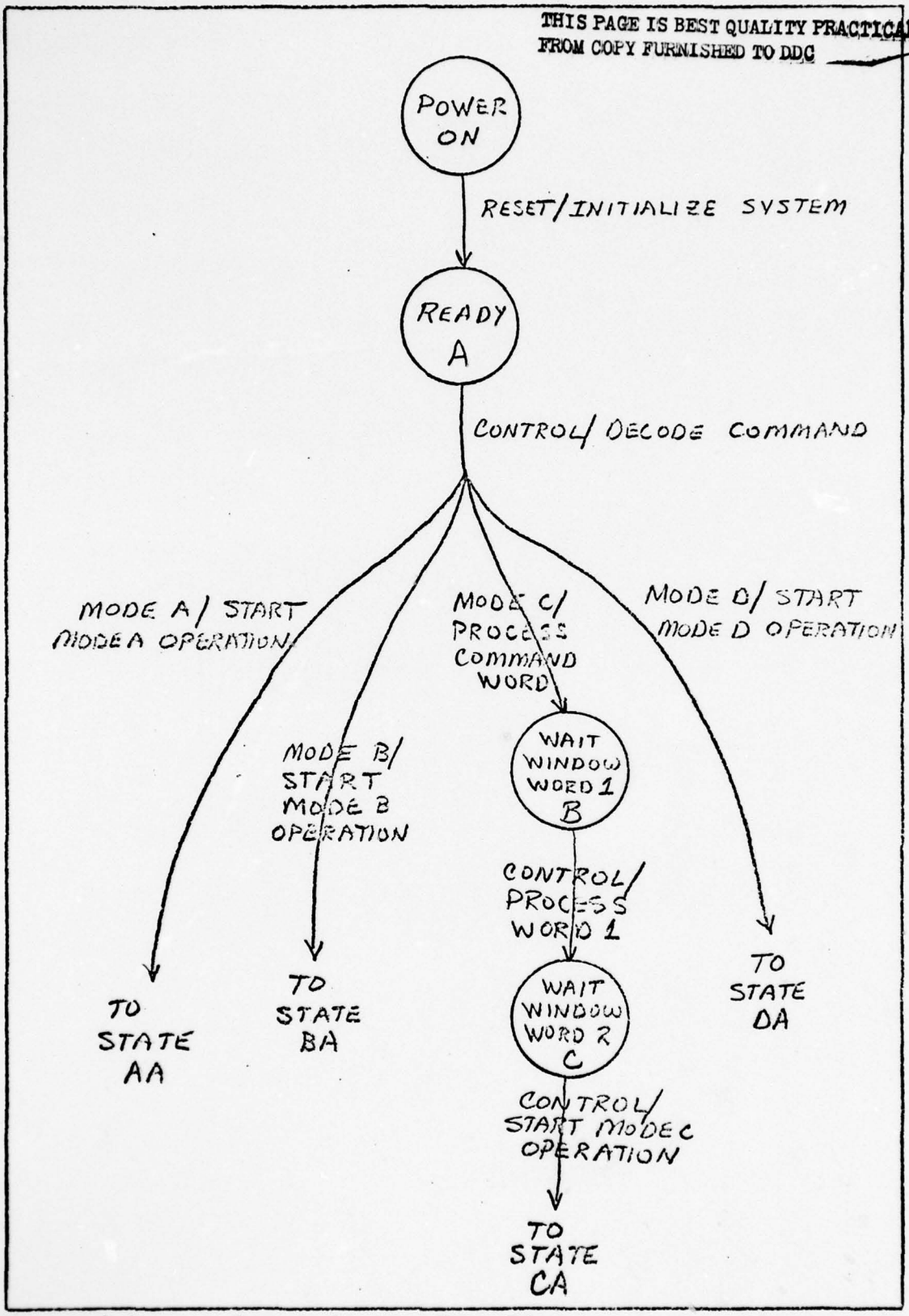


Figure 6-1 System Start-up State Diagram

which transition to make, and this is the reason for the branched transition leaving READY (state A). The branched transition is a departure from normal state diagrams, but it accurately reflects the need for decoding before the proper transition can be made. After decoding, the system sets up for operation in the selected mode, or in the case of MODE C, it waits to receive the nominal time window. The time window comes in two parts of 16 bits each and requires two more interrupts. Once the system is ready for operation, it enters a state specific to the selected mode; these states are indicated by two letter labels with the first letter giving the mode of operation.

The state table for system start-up is presented in Figure 6-2. Across the top of the table are the possible interrupts and their decoded meanings when necessary. The number by the interrupts is the request line on the 8259 Programmable Interrupt Controller assigned to the interrupt. Zero is the highest priority and seven is lowest when more than one type of interrupt appears on a state table. The state table for system start-up shows only three states and no RESET input. This is done to simplify the figure since it is assumed that any time the RESET button is pushed, the system will initialize itself again and return to the READY state.

The entries in the body of the table require special explanation. Each entry is in two parts: the first part gives the next state for the given interrupt, and the second part is a code to identify the processing accomplished during the transition. An example can show how the code

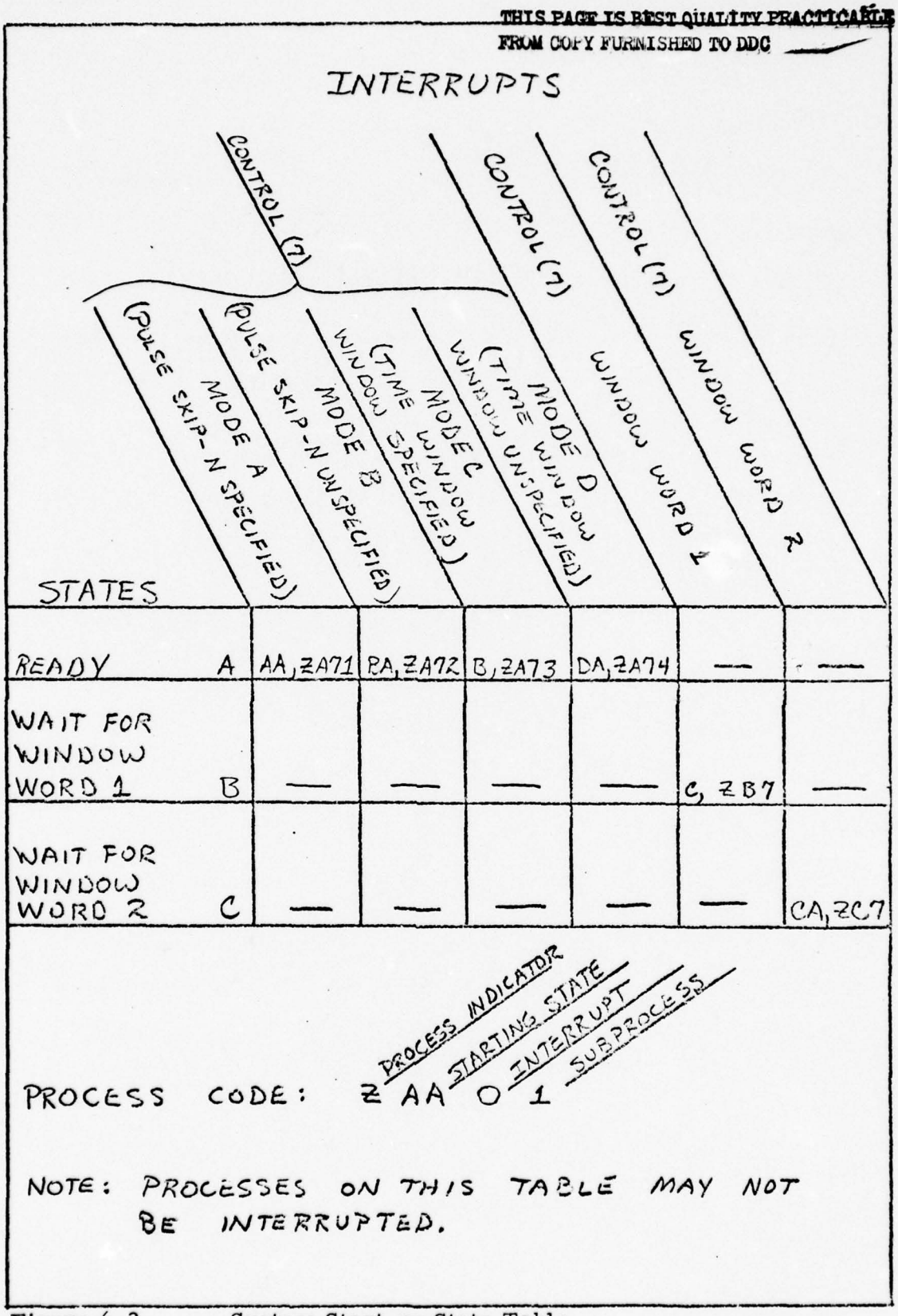


Figure 6-2 System Start-up State Table

is derived. The first process code in the table is ZA71; the meaning of its various parts are as follows:

Z	General designator for all processes
A	One or two letter identifier for the starting state of the transition
7	The number of the interrupt causing the transition
1	A number assigned to differentiate between different subprocesses which can be performed after an interrupt has been decoded

A dash in the state table indicates an impossible or "don't care" condition. Two more notes about the state table need to be added. The process code assigned to a RESET input is ZR. Finally, all of the processes listed in Figure 6-2 are not to be interrupted. Interrupts can be enabled only after the process is complete.

Data Collection. The general activity for data collection is the same in all four modes, so only one state diagram and one state table are used to describe this activity. In Figure 6-3, the states are labeled XA, XB, etc. where X can be for mode A, B, C, or D. The difference between each mode is in the processes performed during state transitions. To simplify the diagrams and to avoid confusion between different modes, the transition processes are not given on Figure 6-3. However, any process can be easily located on the corresponding state table which is given in Figure 6-4. For example, on the state diagram (Figure 6-3) a $\overline{\text{CONTROL}}$-CONTACT BROKEN interrupt in state XA causes a transition to state XB. In the state table

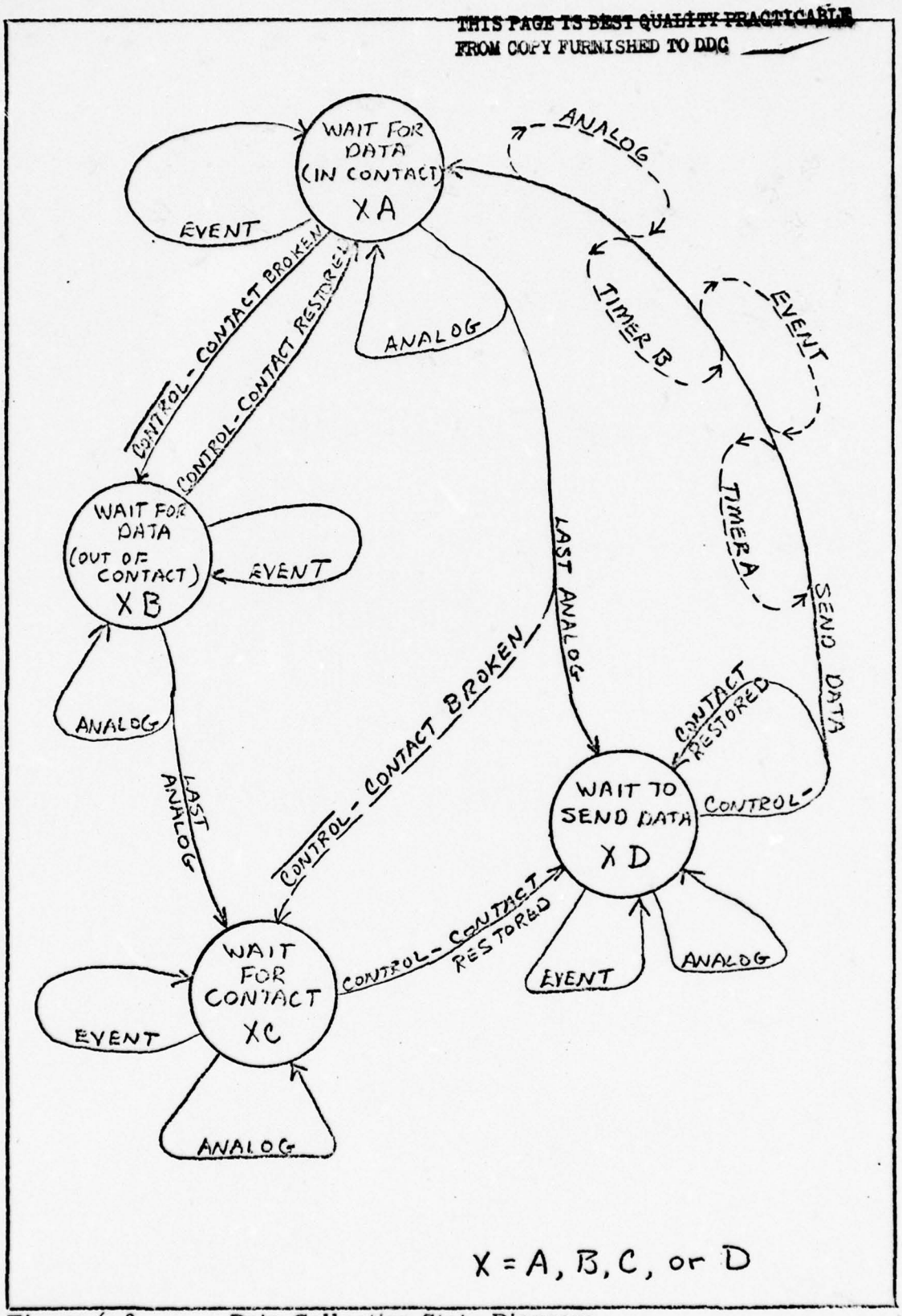


Figure 6-3 Data Collection State Diagram

STATES	CONTROL (0) CONTACT BROKEN	ANALOG (1) NORMAL A/D CONVERSION COMPLETE	EVENT (2) LAST ANALOG DATA TO COMPLETE RECORD	TIMER A (5)	TIMER B (6)	CONTROL (7) CONTACT RESTORED	CONTROL (7) SEND DATA	CONTROL (7) CHANGE MODE		
WAIT FOR DATA (IN CONTACT) XA	XB, ZXA0	XA, ZXA11	XD, ZXA12	XA, ZXA2	—	—	—	—	A, ZXA7	
WAIT FOR DATA (OUT OF CONTACT) XB	—	XB, ZXB11	XC, ZXB12	XB, ZXB2	—	—	XA, ZXB71	—	A, ZXB72	
WAIT FOR CONTACT (FILE COMPLETE) XC	—	XC, ZXC1	—	XC, ZXC2	—	—	XD, ZXC71	—	A, ZXC72	
WAIT FOR SEND DATA SIGNAL XD	—	XD, ZXD1	—	XD, ZXD2	—	—	XD, ZXD71	XA, ZXD72	A, ZXD73	
PROCESSES										
LAST ANALOG DATA FOR RECORD ZXA12	XD→XC, ZXA12→ZXB12	—	—	—	—	—	—	—	—	
SEND DATA ZXD72	—	ZANLG	—	ZEVNT	ZTMRA	ZTMRB	—	—	—	

Figure 6-4 Data Collection State Table

(Figure 6-4), the transition process is ZXAO which is the upper left entry in the table.

Before the individual transitions are discussed, the unusual features of the state diagram, Figure 6-3, must be pointed out. ANALOG interrupts in states XA and XB or a CONTROL interrupt in state XD have branched transition arrows. These are again situations where decoding of the interrupt is necessary. For the ANALOG interrupts, the decoding consists of determining if the analog data completes a data record. This is called the LAST ANALOG interrupt. Each CONTROL interrupt occurs when the minicomputer raises the CONTROL signal. In Chapter IV, it was mentioned that each CONTROL interrupt must be accompanied by an I/O coordination word in I/O port A which specifies why the CONTROL signal is high. The three possible reasons for a CONTROL interrupt are to send data, to restore contact, or to change the mode of operation. The time digitization system must decode the coordination word and this explains the branched CONTROL transition from state XD (the change mode transition is omitted to simplify the diagram). The last unusual feature of the state diagram is the use of broken lines for special transitions. Each special transition occurs while another interrupt is being serviced. In one case, the final state of the transition is changed. In the remaining instances, the current processing is suspended for a higher priority routine. Note that the special processes are not interruptable so only two levels of interrupts are allowed in the system. Next, the individual transitions

are described in more detail.

Wait for Data, XA, is entered upon completion of system start-up. In this state the minicomputer is still "in contact" with the time digitization system, that is the CONTROL flip-flop is still high so interrupts to the HP 2100 are still possible. State XA should be the most frequently occupied state. It is here that the majority of the EVENT and ANALOG interrupts are expected; during each interrupt, data is read from the special purpose hardware and saved in the RAM, and the system returns to state XA. Any $\overline{\text{CONTROL}}$ interrupt and an ANALOG interrupt which provides the last data to fill a data record cause a transition to new states. The $\overline{\text{CONTROL}}$ interrupt causes a transition to XB, a state in which the HP 2100 can not be interrupted when a data record is completed and ready for transmission.* From state XA, an ANALOG interrupt which completes a record causes an interrupt request to the HP 2100 and a transition to XD to wait for a response from the minicomputer. The transition from XA to XD may be interrupted by $\overline{\text{CONTROL}}$ in which case the final state becomes XC. This is to eliminate a possible ambiguity in the communication between the HP 2100 and the time digitization system.

If the LAST ANALOG interrupt service routine is not allowed to be interrupted, it is possible for CONTROL to go low and return high

*One reason for the HP 2100 to clear CONTROL for a period of time is that with several time digitization systems in operation, the one with highest priority may block data transmissions from lower priority systems. As each system has separate CONTROL flip-flops, each can be disabled separately.

before the LAST ANALOG service routine attempts to signal a data record is ready by setting the FLAG. The I/O protocol mentioned earlier requires that the HP 2100 provide an I/O coordination command word each time it sets CONTROL (except during data record transmission). Suppose that while a LAST ANALOG interrupt is being processed, the HP 2100 clears CONTROL, loads an I/O coordination word to start a change in the mode of operation, and then sets CONTROL. If the LAST ANALOG service routine sets the FLAG at this point, the HP 2100 will interpret the signal as an acknowledgement of its command rather than a request to transmit a data record. The ambiguity is avoided by allowing a $\overline{\text{CONTROL}}$ interrupt at one point during a LAST ANALOG service routine just before the FLAG is set. If $\overline{\text{CONTROL}}$ occurs, then the FLAG is not set and the process terminates in XC. Interrupts must be disabled while the FLAG is set or other complications can arise so a restriction must be applied to the HP 2100. The restriction is that CONTROL must remain low for a minimum of 16 times the basic 8080A-1 cycle time (5.12 microseconds at the maximum .32 microsecond clock rate). This allows the time digitization system to disable interrupts, set the FLAG, and transition into state XD to wait for a CONTROL interrupt. If it happens that the HP 2100 clears CONTROL while the 8080A-1 interrupts are disabled, the time restriction assures that CONTROL remains low until after the attempt to set the FLAG. The FLAG can not be set with CONTROL low, but it does not matter because in XD, a CONTROL interrupt will occur for one

reason or another, and the I/O coordination word demanded by the protocol will remove any uncertainty as to what actions are to be accomplished.

Turning now to state XB where the time digitization system operates "out of contact" with the HP 2100, EVENT and ANALOG interrupts are handled normally without any special multiple level interrupts. A CONTROL interrupt returns the system to state XA while a LAST ANALOG interrupt causes a transition to state XC. In XC, data continues to be collected as long as storage is available. A CONTROL interrupt which restores contact causes an immediate request to send a data record and a transition to XD.

State XD is primarily for waiting for a CONTROL interrupt. Both EVENT and ANALOG interrupts have priority and are serviced as they occur, but to get to XD, an attempt to signal a data record ready must have been made. Consequently, at a CONTROL interrupt, data transmission is initiated providing the command to send data is received. If the I/O coordination word indicates that contact is being restored, then another attempt to set the FLAG is made and XD is reentered.

The last transition of interest follows a CONTROL interrupt with a command to send data. Once the data transmission process starts, it may be interrupted four ways: the two I/O timers may stop transmission temporarily until the HP 2100 catches up to the time digitization system, or EVENT and ANALOG interrupts may preempt sending the data record. When a complete data record is sent, the system returns

to state XA. Following the transmission of the last data word, the HP 2100 must clear and reset CONTROL to avoid causing a TIMER interrupt.

One of the I/O coordination words used with a CONTROL interrupt indicates the mode of operation is to be changed. This interrupt is not shown explicitly on the state diagram but it may occur at any time and cause the system to stop operation and return to state A. In state A (READY in the start-up phase) the system would wait for more commands to define the new mode.

The state table of Figure 6-4 repeats the information of the state diagram and adds the process identification codes. In the left column, below the states, are the two processes which are allowed to be interrupted. The table entry for process ZXA12 shows the change of process and the new final state that is caused by a $\overline{\text{CONTROL}}$ interrupt. The $\overline{\text{CONTROL}}$ interrupt is allowed only after the analog data has been read into the processor so a return to the LAST ANALOG service routine is not necessary (but the stack pointer must be changed). The entries for ZXD72 (SEND DATA) are special codes for processes which are the same no matter which mode of operation the system is using. These processes always return to the SEND DATA service routine.

This completes the software structure for the time digitization system. Before the procedures for developing flow charts and code are discussed, some details about the data formats used for transmission between the HP 2100 and the time digitization system should be presented.

Data Formats

The data formats used for communication between the HP 2100 and the time digitization system naturally fall into two classes, the formats for commands sent by the minicomputer and the format for the data samples which the time digitization system sends to the minicomputer. The command words give the mode of operation, the selected event, and the option parameters if they are specified. A special command called the I/O coordination word provides a code to give the purpose for each CONTROL interrupt. The data sample format is standard for all modes and options which should simplify the development of analysis programs. In the next few paragraphs the specifics of the formats are presented.

Minicomputer Commands. When the time digitization system is in the READY state, the first thing necessary to start operation is the command word shown in Figure 6-5a. The eight bits loaded into I/O port A tell the option to be used, the selected event or events, and if an option parameter is specified. If a bit is set to a one, then the corresponding condition is selected or true. The order of the bits in this part of the command word matches the order used in the command register of Subsystem 2. When the pulse skip option is selected and the option parameter is specified, N is placed in I/O port B. The value for N is assumed to be an eight bit binary number so no conversion by the time digitization system is necessary. In all but MODE C, operation can begin immediately after the command word is received.

In the case of MODE C, a 32 bit time window is necessary. This

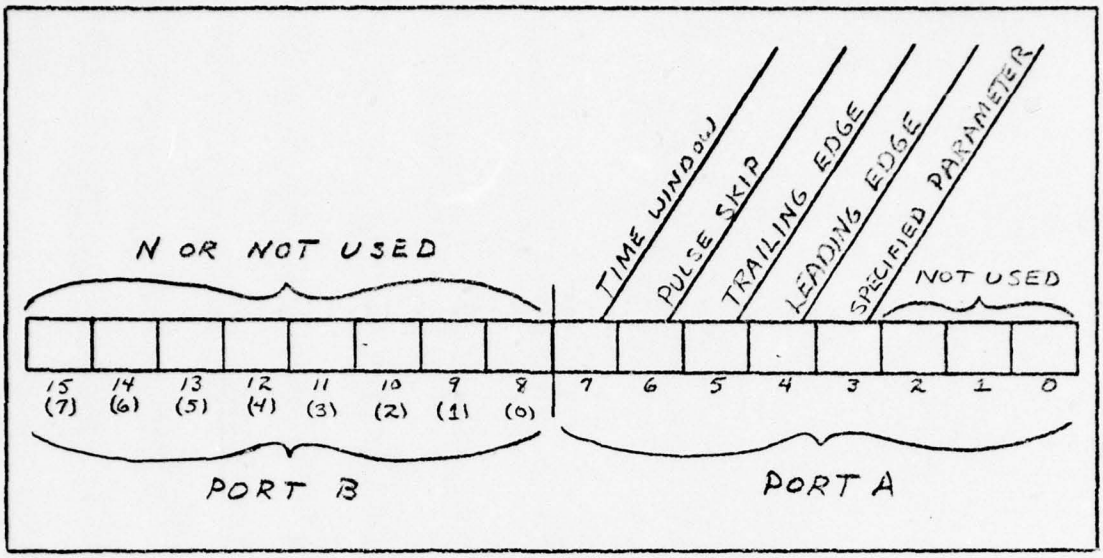


Figure 6-5a Minicomputer Command Word

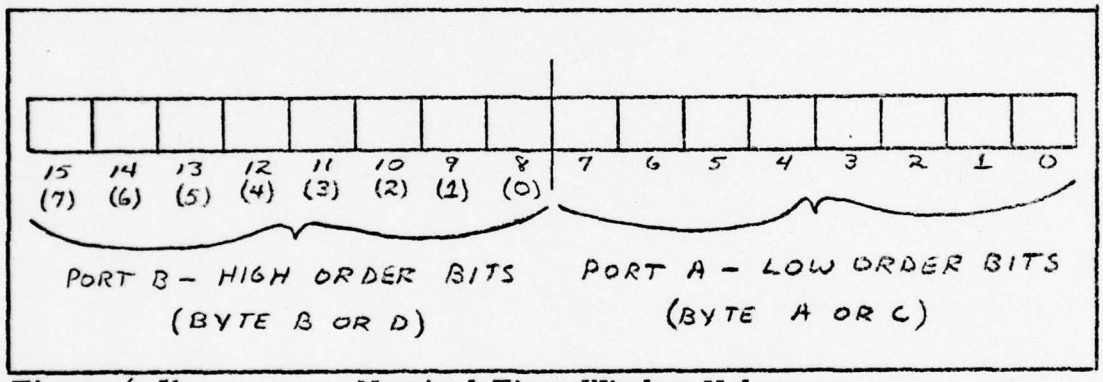


Figure 6-5b Nominal Time Window Value

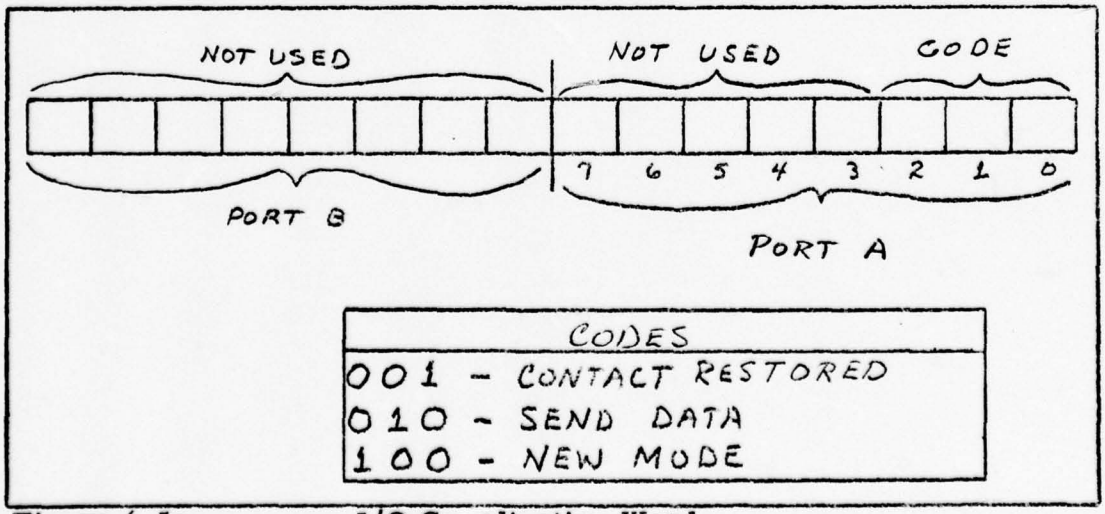


Figure 6-5c I/O Coordination Word

is divided into two 16 bit command words, each of which uses the format of Figure 6-5b. The first word contains the low order bits of the window, and the second word contains the high order bits. For each word, port A should receive the low order byte. To simplify the operation of the time digitization system, the HP 2100 should convert the window length in seconds which is provided by the operator to a binary value which is the number of 25 ns periods contained in the window length. This number can then be directly loaded into the time window registers (after 2 is subtracted to compensate for the timer hardware delay).

Finally, any time the HP 2100 sets CONTROL, except during transmission of a data record, an I/O coordination word must be provided. Three coordination codes are used here: 001 for contact restored; 010 for send data; and 100 for change mode of operation. Another code for resetting the hardware and maintaining the same mode may be desired. Figure 6-5c shows the placement of the codes.

Data Samples. The requirements definition specified that a data sample may contain 16 words of 16 bits, and the system is designed to process data samples of that size at the required speed. However, samples can be organized into only 12 words or 24 bytes as follows:

Byte 1	Selected option; selected event; parameter specification; channel identification (see Figure 6-6a for format)
Byte 2	N or nominal time window byte A
Bytes 3-5	Not used in pulse skip option; nominal time window bytes B to D

Bytes 6-10	Event time (time digitization value); status in last four bits (see Figure 6-6b for format for byte 10)
Bytes 11-12	Net count
Bytes 13-24	Analog data

Odd numbered bytes are loaded into I/O port A and even numbered bytes go into port B. With this format for a data sample, it is apparent that the time digitization system can exceed the required data transmission rate. It may be desirable, however, to use two bytes for each 12 bit A/D converter output, and to separate some of the information in the first byte into different bytes to simplify data analysis. This could increase the data sample size up to the 16 word or 32 byte limit.

In the 24 byte format given here, information in bytes 1 through 5 should be constant for each data record. The remaining bytes are variable for each data sample. Thus in a data record, 608 bytes of storage are required for the variable data, and five more bytes are necessary for the constant data. Each additional record which is stored by the time digitization system requires another 613 bytes of RAM. This explains the need for the 1280 bytes of RAM storage given in the Subsystem 1 hardware selection. In the next section, the ROM requirement estimate is explained along with some programming conventions which depend on the formats established for the HP 2100 commands.

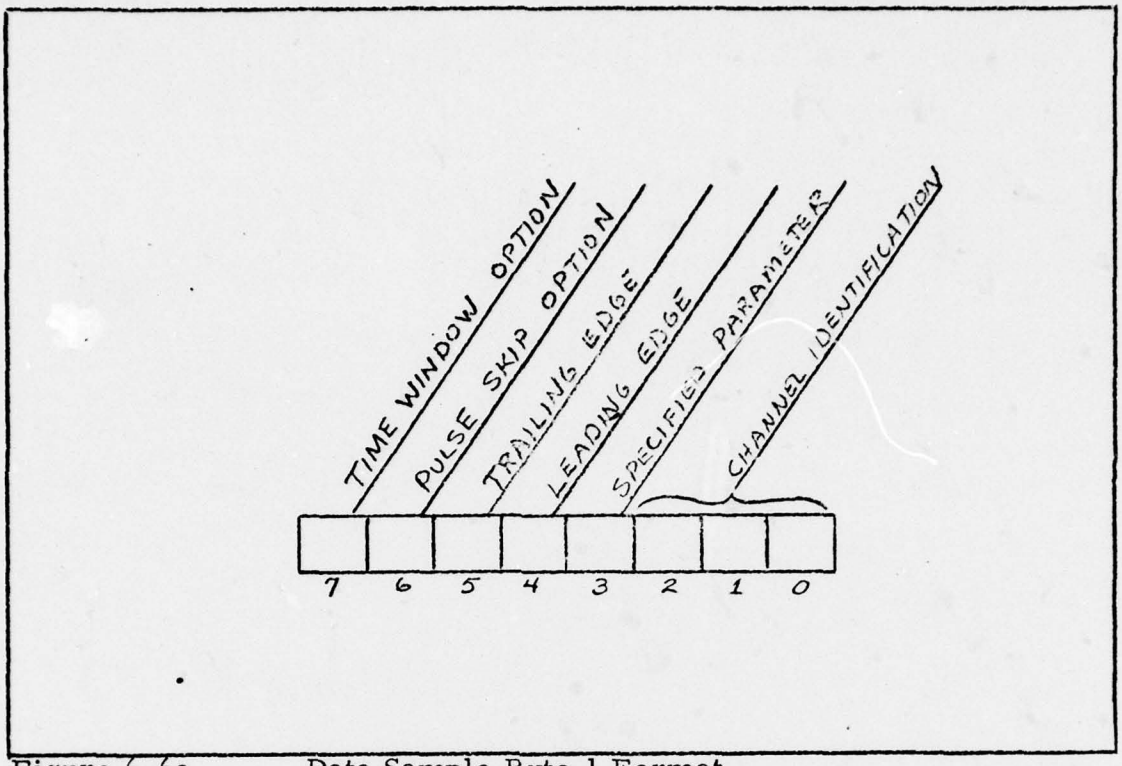


Figure 6-6a Data Sample Byte 1 Format

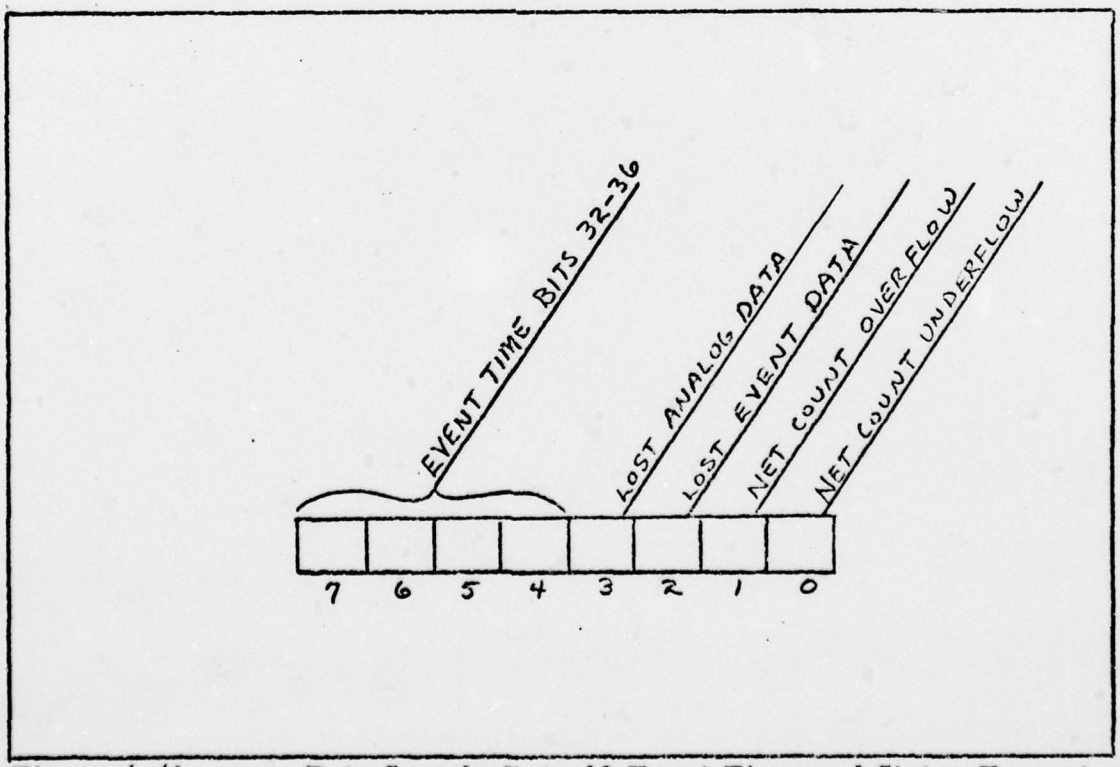


Figure 6-6b Data Sample Byte 10 Event Time and Status Format

Flow Charts and Coding

The information presented in this section does not provide a comprehensive treatment of the coding requirements for the time digitization system. Since software requirements are still subject to change, a selection of the major considerations in writing code for the system is given along with a few typical examples.

The most significant feature of the software is the need for processing interrupts. The PIC allows interrupt traps to be located anywhere in memory and to be four or eight bytes long. It is suggested that the four byte option be used, and that the traps be located in the ROM. As an example of how to construct an interrupt trap, here is the code for an EVENT interrupt:

```
LHLD EVENT; Load the word in location EVENT into HL
PCHL        ; Move HL into the Program Counter
```

It should be evident that the location labeled EVENT is a pointer in RAM which contains the address of the service routine for an EVENT interrupt. The set of pointers for all of the six possible interrupts constitutes the state table for the system. Changing the pointers changes the current state of the system.

A part of the transition from one state to another is the changing of the interrupt service routine pointers. In some cases, where interrupts are allowed during transition processes, the pointers should be changed at the beginning of the process. Otherwise, at the end of each process, the pointers can be set for the state in which the transition is to terminate.

State transitions should always terminate in a halt condition to await the next interrupt. Recalling that the PIC uses a subroutine CALL instruction to start an interrupt service, a return to a halt condition simply requires a RET or return instruction. For this to work correctly, the instruction following HLT must be a jump back to HLT:

```
HLT     ; Halt
JMP -1  ; Jump back one
```

A number of other general considerations must be kept in mind when code is written for the time digitization system. Both the PIC and the PPI require initialization, and the PIC must be cleared by a special End of Interrupt (EOI) instruction before it can process additional interrupts. An IN or OUT instruction to I/O port B is the method for acknowledging data from the HP 2100 or signalling to the minicomputer that data is ready for reading. At times, for example in the MODE B command word, port B does not contain data. In this case, an IN IOPTB instruction is still necessary to set the FLAG. Finally, the I/O protocol which demands an I/O coordination word for each CONTROL interrupt requires that the PPI be in the receive mode when not being used. Once a send data command is received, the PPI can be changed to the transmit mode.

The manual RESET is a special type of interrupt which causes the 8080A-1 to execute code beginning at location zero. The process associated with RESET is labeled ZR, and Figure 6-7 gives an example of

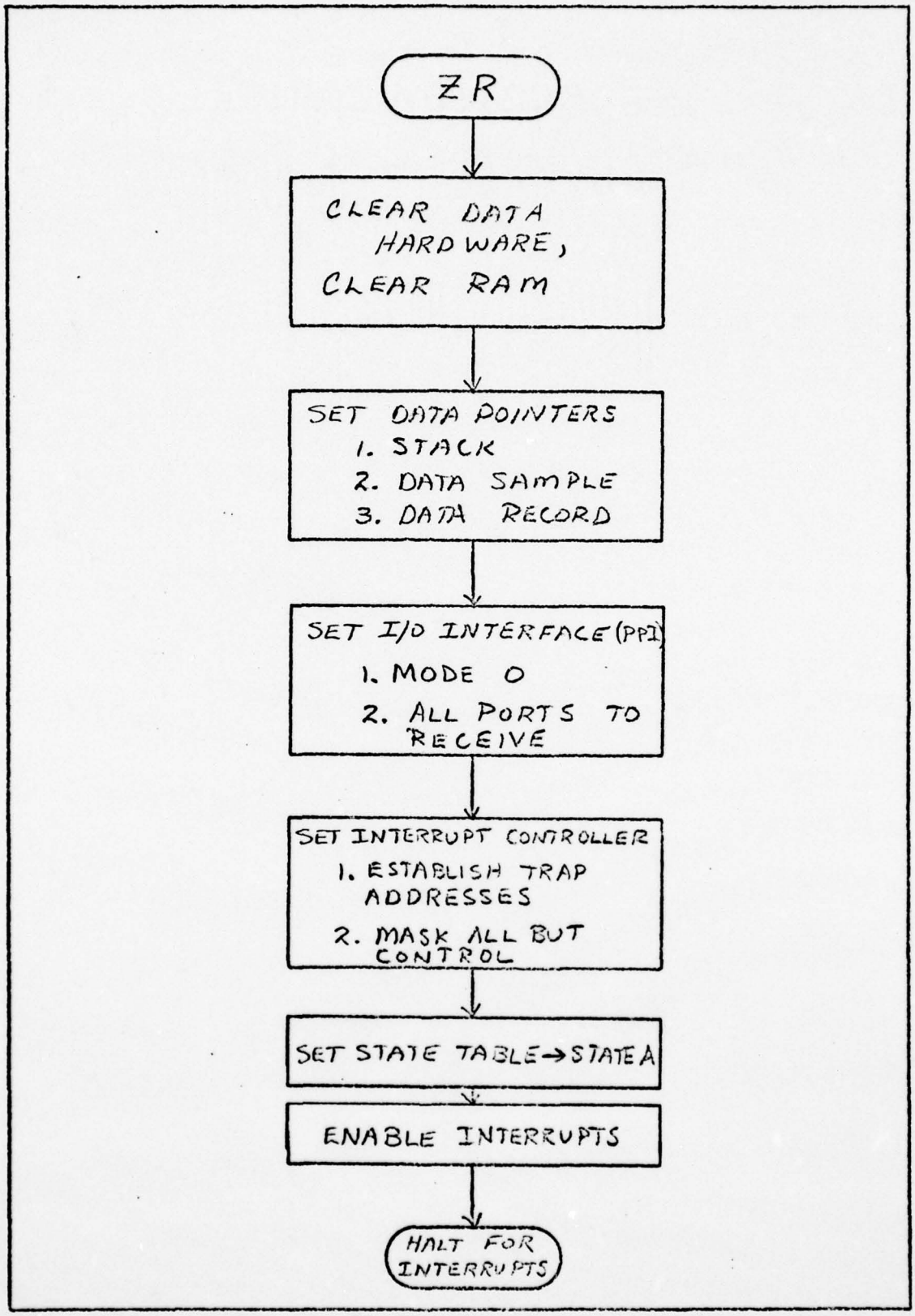


Figure 6-7 Process ZR Flowchart

the activities which must be performed to prepare the system for operation.

Process ZR terminates in Ready, state A, and the next interrupt which the time digitization system expects is a CONTROL interrupt for a command word to start one of the four modes of operation. Figure 6-1 shows a branched transition from state A which is reflected in the flow chart of Figure 6-8. The process of the transition begins as ZA7 and becomes one of four possible subprocesses after the mode has been decoded. In Figure 6-9, the subprocess ZA71 for starting MODE A is given as a representative of the four subprocesses. The command byte referred to in the flow chart is the command from the HP 2100 with a run bit set so the special purpose hardware can operate.

As the state tables show, operations during data collection are more complicated, although the individual processes are not difficult. Four examples are given here which involve $\overline{\text{CONTROL}}$, EVENT, ANALOG, and CONTROL interrupts. A $\overline{\text{CONTROL}}$ interrupt during a wait state causes a process (ZXAO) which simply changes the state table. The remaining interrupts are more interesting.

Processing an EVENT interrupt is similar in all states. Since gathering event data is the primary activity of the time digitization system, interrupts should be left disabled for the duration of the interrupt service routine. Figure 6-10 shows what must be done for process ZAA2 which follows an EVENT interrupt when the system is in state AA. No change of the state table is necessary because an

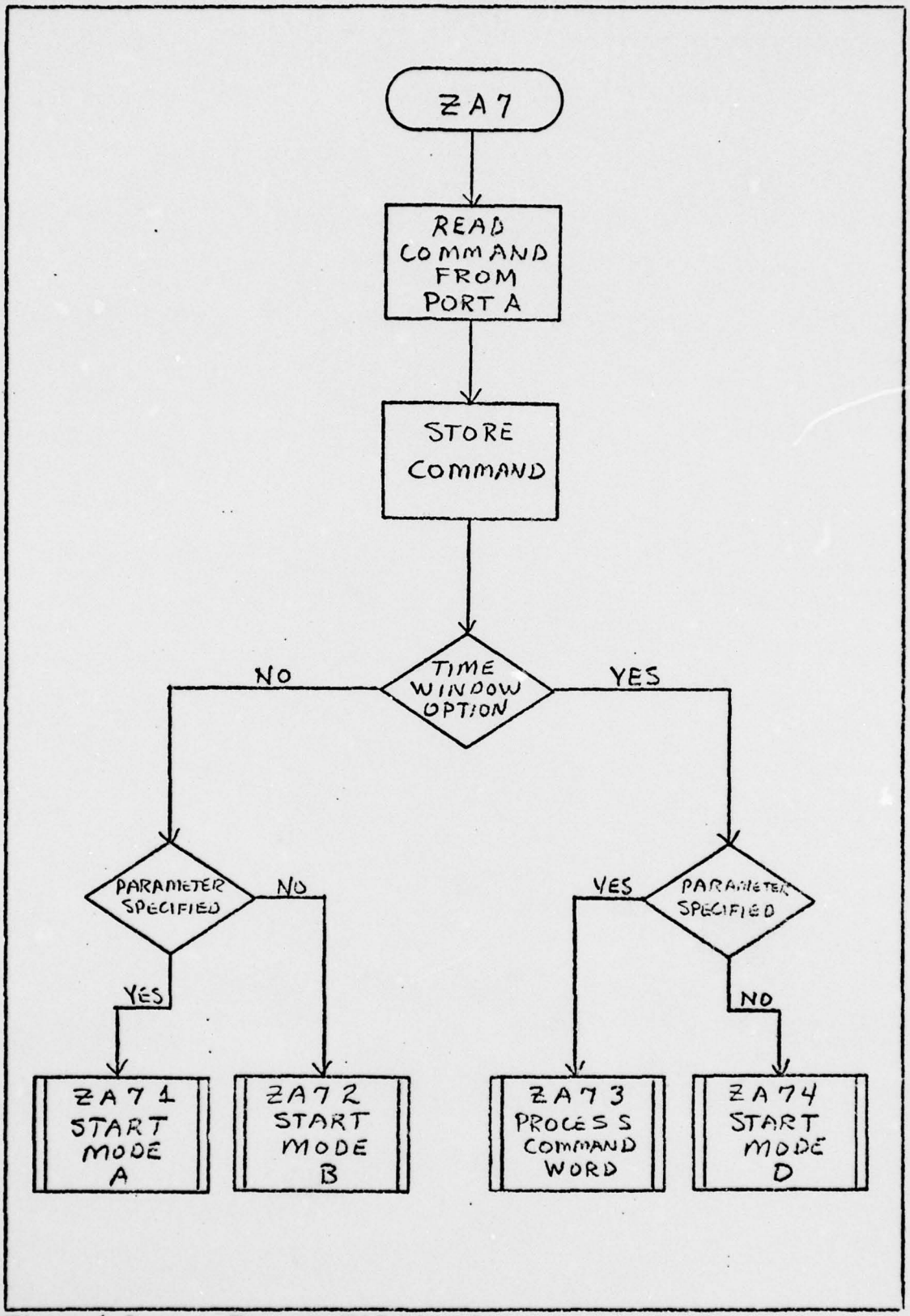


Figure 6-8 Process ZA7 Flowchart

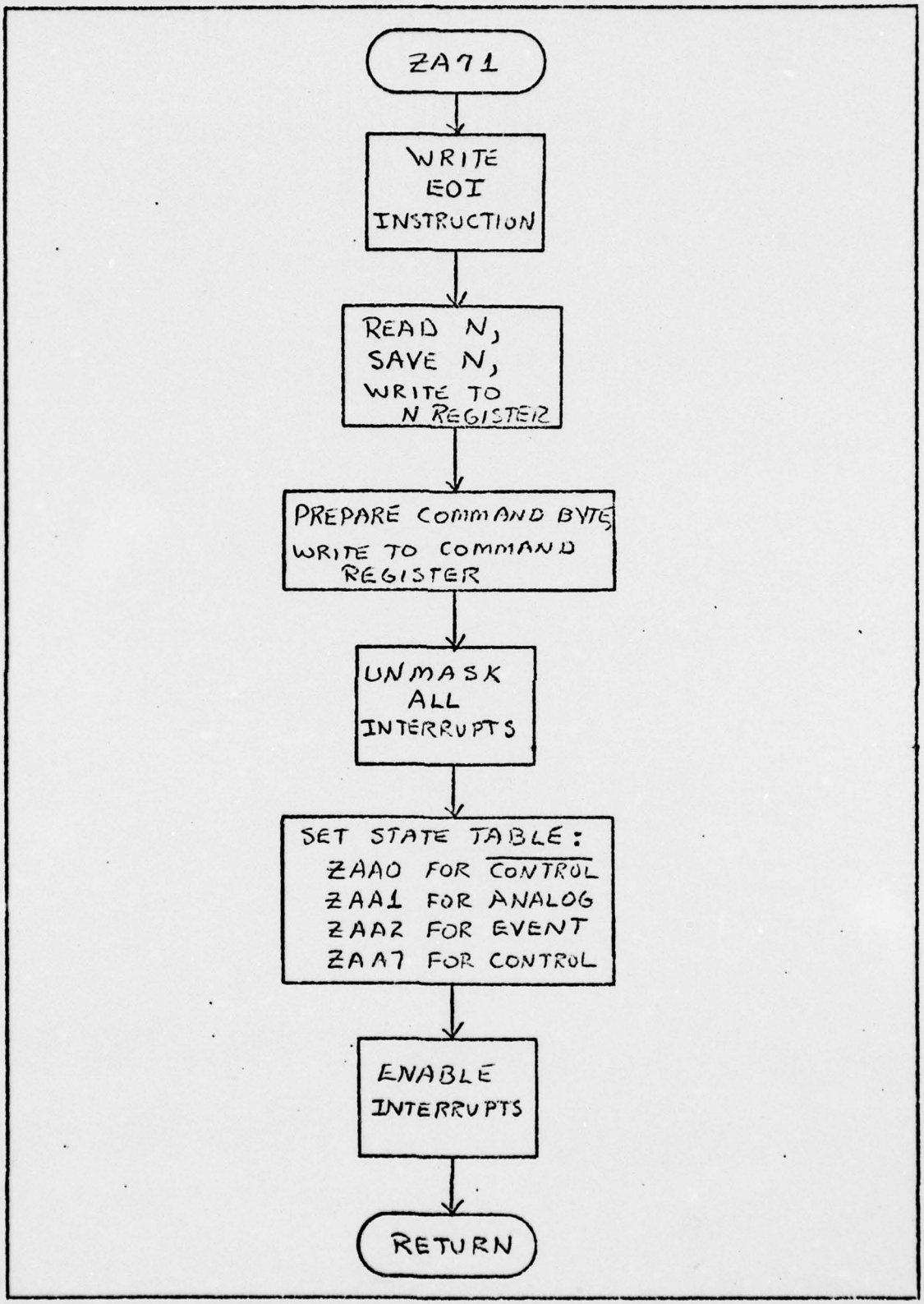


Figure 6-9 Subprocess ZA71 Flowchart

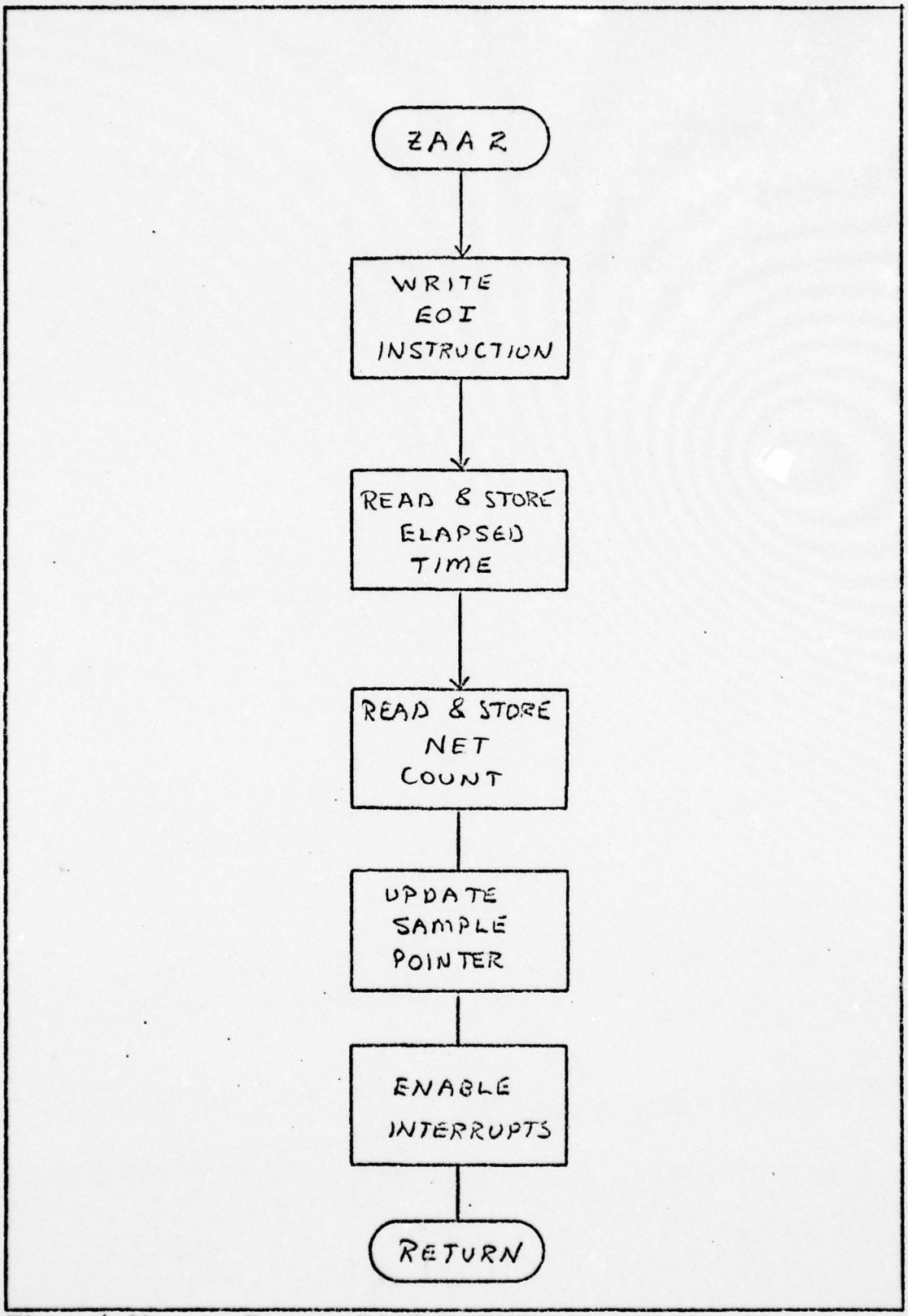


Figure 6-10 Process ZAA2 (EVENT Interrupt) Flowchart

EVENT interrupt never causes a state change.

It is during the processing of ANALOG interrupts that a check should be made to determine if a data record is being completed. Figure 6-11 shows that interrupts need to be enabled while processing such a "LAST" ANALOG interrupt as mentioned earlier. Note how this flow chart compares with the ANALOG interrupt for state XA in the state diagram and state table of Figures 6-3 and 6-4.

In MODE B or MODE D, a LAST ANALOG interrupt is the appropriate time to examine the data rate and to change N or the nominal time window if necessary. For these two modes, the flow charts corresponding to Figure 6-11 would have additional boxes for determining the data rate and updating N or the time window.

Figure 6-12 shows the processing of a CONTROL interrupt in which the I/O coordination word is to send data. In the block labeled Write Complete Data Sample, instructions are necessary to write each of the bytes of data which are constant for the record. Then the instruction sequence presented in Chapter IV is repeated once for each byte of variable data in the sample. Using the 24 byte format presented here, the sequence would be repeated 19 times. A loop is used to send all 32 data samples in a record, but this is the only loop necessary.

Of all the flow charts presented in this section, not one is noticeably complex. Although the software structure contains a large number of modules, or separate transition processes, the coding for each one is relatively simple, and many opportunities exist to use common

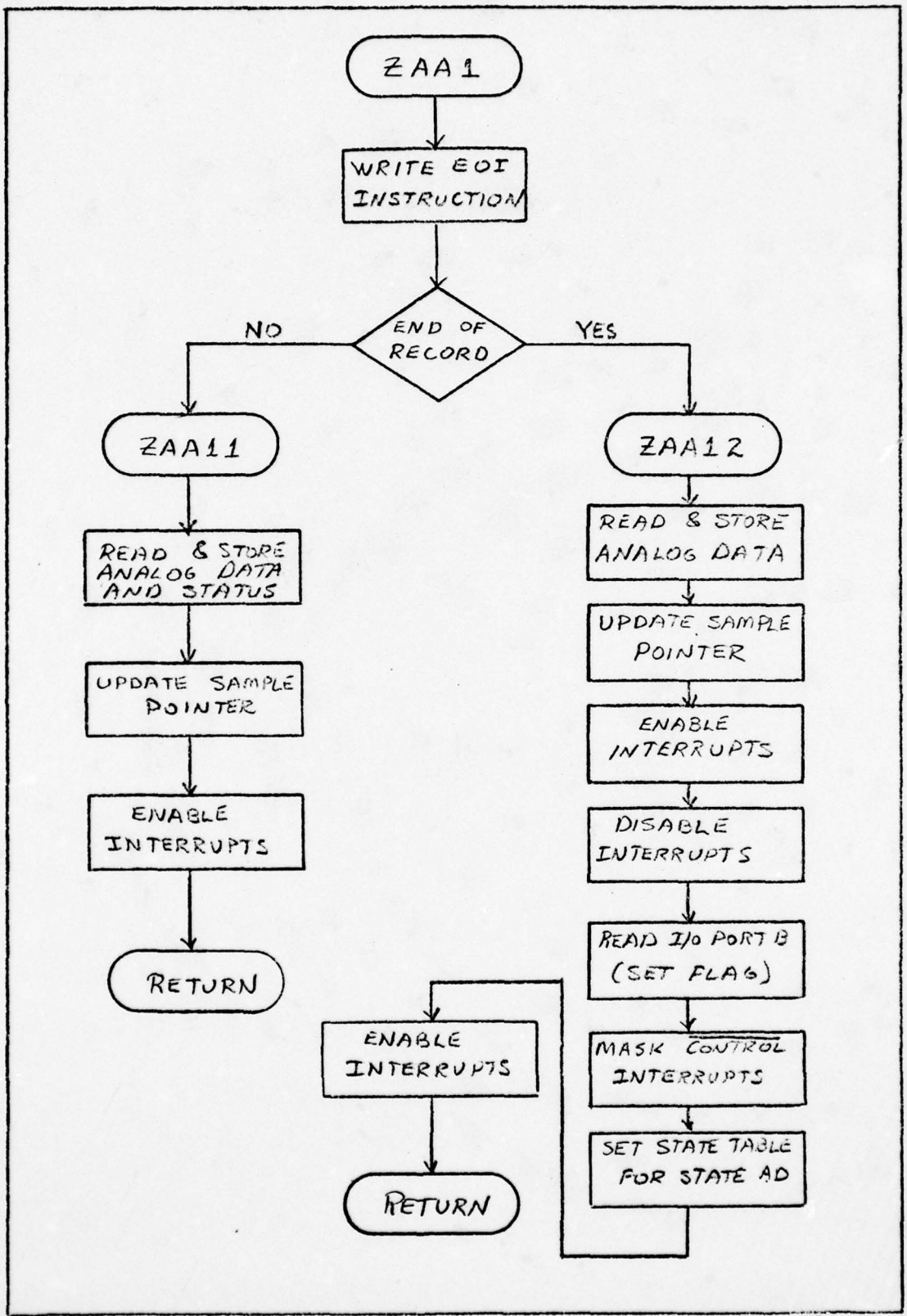


Figure 6-11 Process ZAA1 (ANALOG Interrupt) Flowchart

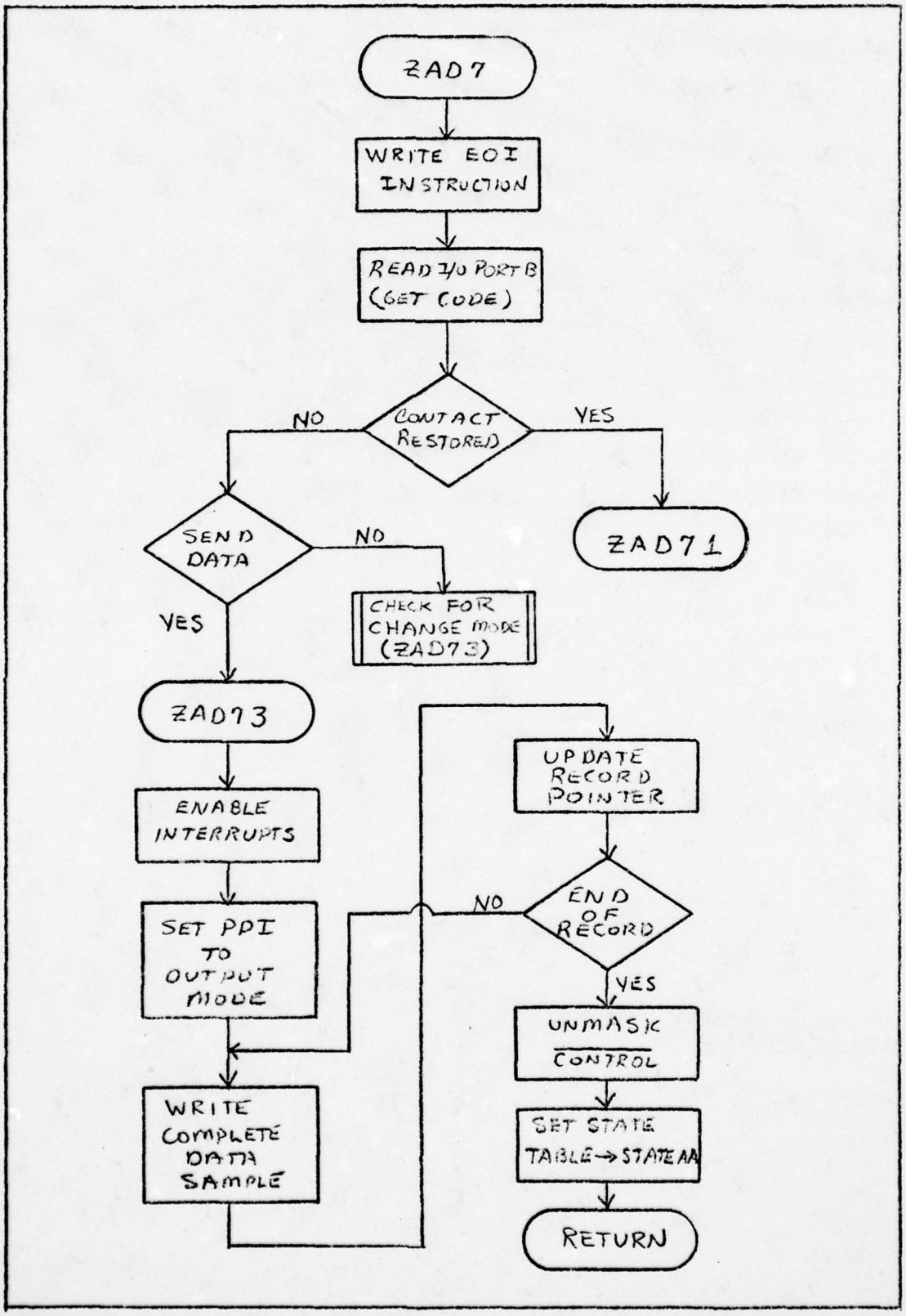


Figure 6-12 Process ZAD7 (CONTROL) and Subprocess ZAD73 Flowchart

subroutines. This is an indication that the state table approach for developing the software structure is an appropriate choice.

One factor which was not considered in developing the software for the time digitization system was transmission error identification and recovery. Under some laboratory conditions this may be very important. Another design decision is to determine how critical transmission errors can be and the sophistication necessary in error processing.

Finally, there is the matter of estimating ROM requirements. In a preliminary coding of the start-up phase, an average of 63 bytes were used for each process (382 total). Assuming that nearly all of the code in the collect data phase is common to each mode of operation, the average of 63 bytes gives a requirement for approximately 1600 bytes for data collection. The system total is then about 2000 bytes, and the closest power of two is 2048. This is at best only an estimate, but by planning to use ROM external to the processor chip, no great difficulty should arise.

The purpose of this chapter was to develop a software structure for the time digitization system, and to present some important factors which must be considered when code is written. The software structure was developed using techniques borrowed from finite state automata theory. These techniques worked because the system is interrupt driven, and because only two levels of interrupts are ever allowed at one time. The chapter also included formats for data which is transmitted between the time digitization system and the minicomputer. The data formats,

the software activities specified in the Subsystem 1 design model, the processor hardware, and the special purpose hardware all impose requirements for coding the modules of the software structure.

The software structure and code complete the portion of the design process which this thesis was to consider. The next chapter summarizes the results of the thesis and makes recommendations for continuing the design effort.

AD-A055 228 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN OF A LABORATORY DATA ACQUISITION SYSTEM (TIME DIGITIZATI--ETC(U)
MAR 78 J R MANEELY

UNCLASSIFIED AFIT/GCS/EE/78-4 NL

3 OF 3
AD
A055228

END
DATE
FILMED
7-78
DDC

VII Results and Recommendations

The primary objective of this thesis was to verify that a time digitization system is feasible by developing a design for the system. The second objective of the investigation was to determine the effectiveness of several high level design tools on a combined hardware and software design project. It is concluded that with the present performance specifications the time digitization system is feasible. The reasons for this conclusion are presented in the next section. The second section of this chapter discusses the design method results. It is unreasonable to claim that one design method is best for all designers for a particular application, and no such claim is made here. However, the results of the design tools, particularly the techniques patterned after SADT, show these tools can be very successful and should be considered by designers when planning a design project. The final section of this chapter presents recommendations for continuing the design of the time digitization system. Some major design decisions are yet to be made, but once the decisions are made and the design documents updated, the time digitization system can be constructed.

Design Results

The conclusion that the time digitization system is feasible depends on the results of the processor design, the special purpose

hardware design, and the software structure. Each set of results is considered in the next three paragraphs. Cost considerations were also a part of this investigation and they are also discussed. The section concludes with a comparison of the system performance requirements with the expected performance of this design.

The critical performance factor for the processor design is the data transmission rate. In Chapter IV, it was demonstrated that the Intel 8080A-1 can slightly exceed the required transmission rate when supported by the remaining hardware in the Subsystem 1 design. The implication is that the processor hardware can meet all of the processing speed requirements.

All the special purpose hardware functions (except signal conditioning) which were identified in the Subsystem 2 design can be performed by TTL hardware. This was demonstrated in the first part of Chapter V. Still, to determine the feasibility of the special purpose hardware design, it was necessary to go down to the level of selecting individual chips and specifying their interconnections. The reason for the great amount of detail was the 40 MHz clock which demands performance which is near the limit of TTL Schottky capabilities. However, a check of all signal paths in the Subsystem 2 circuit layout will show that the maximum signal propagation times and flip-flop setup times do not exceed the 25 nanosecond clock period, so the special purpose hardware is feasible.

Finally, feasibility depends on the software structure and the

capability of the processor to execute the software. The software structure was developed to perform the functions of the Subsystem 1 design model which represents the time digitization system software requirements. The structure was also developed to operate with the 8080A-1 interrupt processing system supplemented by other processor system hardware. The net effect is that the processor system can perform the required functions using the software structure.

After feasibility, system costs were also considered. The cost issue is complex; precise figures are virtually impossible to forecast. Through the design process, the lowest cost alternative was selected when options existed. The processor system may cost several hundred dollars, and the special purpose hardware may take 80 to 100 chips at an average of $2.00 per chip. Still, the design decisions which remain can cause wide variations in total cost. Once the decisions are made and the design modifications completed, then costs should be computed and analyzed for acceptability.

It is useful to compare the performance requirements for the time digitization system with the performance expected from the system design presented in this thesis. The design meets all performance requirements as specified in the Statement of Work. In the case of the elapsed time clock, the window timer, and the net count combinational logic, virtually no improvement in performance is possible because of signal propagation time limitations. Thus the 40 MHz clock rate is the fastest possible using TTL Schottky technology. The input

pulse width and frequency are another matter. The slowest of the special purpose hardware functions, the net count counters, stabilize in 200 nanoseconds which is well below the 1 microsecond pulse width minimum promised in the requirements definition. This design will function properly with 200 ns input pulses. Because the event counter and the window timer can operate with values larger than those originally specified, a higher input pulse frequency is possible. However, the 8080A-1 is near the limit of its performance if it is to process 16 word data samples. Using the smaller 24 byte or 12 word data sample format, then the 8080A-1 can exceed the data transmission specifications by a little less then 25%. All of these performance capabilities and limitations should be kept in mind when assessing data acquisition needs for inertial component testing over the next few years, and making cost-capability tradeoff decisions.

Design Method Results

The major design tools employed in this investigation were the digital system life cycle framework, the SADT techniques for requirements definition and system design, and the finite state automata approach to software structure. The results of each tool are described in this section.

The system life cycle outline provided good design discipline. The concept was applied early in the actual design process which kept the design effort organized and complete. A number of design

frameworks exist, and this is perhaps no better than several others. However, without a general design framework, activities in some of the life cycle phases can be slighted, especially the requirements definition and system design. The result can be an unacceptable or much less than optimum design.

Of the tools applied within the various life cycle phases, the ideas derived from Structured Analysis were the most prominent and most successful. In the requirements definition phase, SA provided an organized, methodical approach to breaking the design problem down into its significant functional parts. Because the method was well received by CIGTF personnel, the design process proceeded in the right direction from the beginning. In the system design phase, SA techniques helped bridge the distinct gap that normally exists between functional requirements and optimal implementations. By applying the concept of concurrency to the requirements definition model, a design model was easily developed. All that was necessary was a simple reorganization of the original model. The resulting sets of software functions and special purpose hardware requirements were simple enough so appropriate implementations were easily identified.

Structured Analysis models are commonly used to specify functional requirements for software designs, but the application of SA techniques to a combined hardware and software design is another matter. The system design models developed in Chapter III demonstrate that SA techniques can produce a very effective combined design.

The ease with which a suitable processor system was selected in Chapter IV, and especially the simple, direct conversion of the Subsystem 2 model to MSI and SSI hardware in Chapter V validate the claims about the SA techniques. For combined hardware and software designs, the Structured Analysis Design Technique should be seriously considered.

In the software design, the finite state approach provided small, simple modules. Relationships between modules were well defined by the state diagrams and state tables. Somewhat less well defined were the functions which were to be performed by each module. For this, the SA design model and a knowledge of the hardware were necessary background. The finite state automata design approach is a useful tool but it has limitations.

Recommendations

A list of recommendations to complete the development of the time digitization system is given in the last part of this section. Before the list is presented, however, the major design decisions which remain are summarized.

Four major design decisions remain. The 40 MHz clock rate is the critical factor for determining the complexity and hence the cost for Subsystem 2 hardware. Reducing the clock rate to 10 MHz might reduce the cost of the special purpose hardware by one third by allowing standard instead of Schottky TTL chips. The selection of the method

for interfacing the time digitization system to the HP 2100 can have as profound an effect on the Subsystem 1 or processor system hardware as the 40 MHz clock rate has on Subsystem 2. Selecting DMA, for example, and using more than two interface control signals could reduce the number of interrupts necessary. This would allow using an Intel 8085 microprocessor and would eliminate the Programmable Interrupt Controller (8259), as well as the Clock Generator (8224) and the System Controller (8228). DMA is an expensive interface method, but if it is already available or can be used in other applications, the additional cost may be worthwhile. The last two design decisions are somewhat less important. The number of data records to be accumulated in the time digitization system before the records are transmitted to the HP 2100 determines the RAM necessary for the system. More RAM gives greater programming flexibility for the HP 2100 and raises the system cost. Finally, the amount of error checking done by the time digitization system affects the complexity of the system software and perhaps the ROM requirements.

The recommendations for continuing the design of the time digitization system are listed below in a reasonable order for completion.

1. Make the necessary design decisions.
    a. Determine the necessity for the 40 MHz clock rate.
    b. Select the HP 2100-time digitization system-minicomputer interface.
    c. Determine the number of data records to be stored in the time digitization system.

d. Determine the level of error checking necessary in the system.

2. Modify the design model as necessary to compensate for any changes made in the design criteria.

3. Select new hardware as required for changes in the design model; rework the software structure to include any changes in interrupts and processing requirements.

4. Modify the circuit layout as necessary and develop code.

5. Begin testing and integration.

   a. Assemble and test Subsystem 2 which contains the most critical hardware.

      i. Test the net count logic.
      ii. Test the elapsed time clock.
      iii. Test the window timer and event counter.
      iv. Test the remaining hardware.

   b. Test software using computer simulation if possible.

   c. Integrate Subsystems 1 and 2; test interfaces and combined operation.

   d. Integrate the time digitization system and the HP 2100; test interfaces and combined operation.

6. Document test results and operating procedures.

The design decisions in recommendation 1 should be made strictly on the basis of CIGTF requirements. No designer preference should be inferred from remarks made in this paper. The possibility of substantial design modification should also not be allowed to affect the decisions. The design process steps are well established, and changes can be easily incorporated into each phase. The primary objective of these recommendations as well as the entire paper is to lead to a time digitization system which meets CIGTF needs.

Bibliography

1. 6585th Test Group, Central Inertial Guidance Facility. Guidance Laboratory Time Digitization System. Statement of Work. Holloman AFB, New Mexico: 6585th Test Group, (undated).

2. Rose, C. W. and J. D. Schoeffler. "Microprocessors for Data Acquisition," Instrument Technology, 21: 65-69 (September 1974).

3. Ward, Ann R. "LSI Microprocessors and Microcomputers: a Bibliography," Computer, 7: 35-39 (July 1974).

4. -----. "LSI Microprocessors and Microcomputers: a Bibliography Continued," Computer, 9: 42-53 (January 1976).

5. ASD-TR-76-11. Management Guide to Avionics Software Acquisition, Volume I - An Overview of Software Development and Management. Dayton, Ohio: Logicon, June 1976.

6. Lane, A. "Microprocessor System Design," Digital Design, 5: 62-66 (August 1975).

7. Boehm, B. W. Software Engineering. Redondo Beach, California: TRW Systems Engineering and Integration Division, (undated).

8. 9022-73.2. Structured Analysis Reader Guide. Waltham, Massachusetts: SofTech Inc., May 1975.

9. 9022-78R. An Introduction to SADT, Structured Analysis and Design Technique. Waltham, Massachusetts: SofTech Inc., November 1976.

10. Yourdan, Edward and Larry L. Constantine. Structured Design. New York: Yourdon Inc., 1975.

11. Cushman, Robert H. "EDN's Third Annual Microprocessor Directory," EDN, 21 (21): 44-89 (November 1976).

12. Nichols, A. J. and Kenneth McKenzie. "Microprocessor Basics: Part 2," Electronic Design, 24 (10): 84-92 (March 10, 1976).

13. Intel Corporation. Intel Data Catalog 1977. Santa Clara, California: Intel Corporation, 1977.

14. Texas Instruments Inc. The TTL Data Book (Second Edition). Dallas, Texas: Texas Instruments Inc., 1976.

15. McGowan, Clement L. and John R. Kelly. A Review of Some Design Methodologies. Waltham, Massachusetts: SofTech Inc., 1976.

Appendix A

Structured Analysis Diagrams (Ref 8)

This appendix gives a short description of how Structured Analysis models are constructed and explains the SA diagram conventions used in this paper. It must be noted that the format used to present the models in this paper is not standard according to the rules developed by SofTech. The changes were made to present the models in a manner which is more familiar to readers who have no experience with SA models. Although the format is not that used by SofTech, the diagrams of the models are organized and related according to SofTech procedures, and the conventions used to construct individual diagrams are standard.

The Structured Analysis Design Technique is a general purpose top-down, modular technique for modeling functions. The functions may be as varied as farming or manufacturing, but SA was developed primarily as a software requirements definition and design tool. Although a complete SA model actually consists of two models, one for activities and one for data, this paper employs only activity models so the conventions described here are those which apply to activity models.

An SA activity model consists of a series of diagrams which present in progressively more detail the activities necessary to perform some function. Each diagram represents a self-contained activity which is part of the overall function. A diagram shows how its

activity is decomposed into subactivities, and how the subactivities are related to each other. The subactivities in each diagram may then be decomposed on separate diagrams which leads to a tree structure of several levels. At the top is one diagram which represents the whole function, and at the bottom are the diagrams which show the most detailed activities.

Figure A-1 shows how an SA model would appear if all the diagrams were on one page. Of course, in real SA diagrams only one level of decomposition is shown, but the figure demonstrates the top down nature of SA and the way activities are grouped into modules. In the figure, as in real models, one large box represents the whole function, and that is decomposed into successive levels of related activities. The decomposition process continues until the desired amount of detail has been developed, which may require more levels than shown in Figure A-1. Another thing to note is that while the figure shows only 3 subactivities in each decomposition, any number from 3 to 6 is acceptable.

From Figure A-1, it should be apparent that SA diagrams are constructed with boxes and arrows. In an activity model, each box represents an activity, and is called a node. Arrows represent "data" where the word data is used in a very general sense to include anything that is not an activity. Figure A-2 shows the different meanings given to arrows depending on which side of a box they enter or leave. An input is data that is modified by the activity to produce an output. A

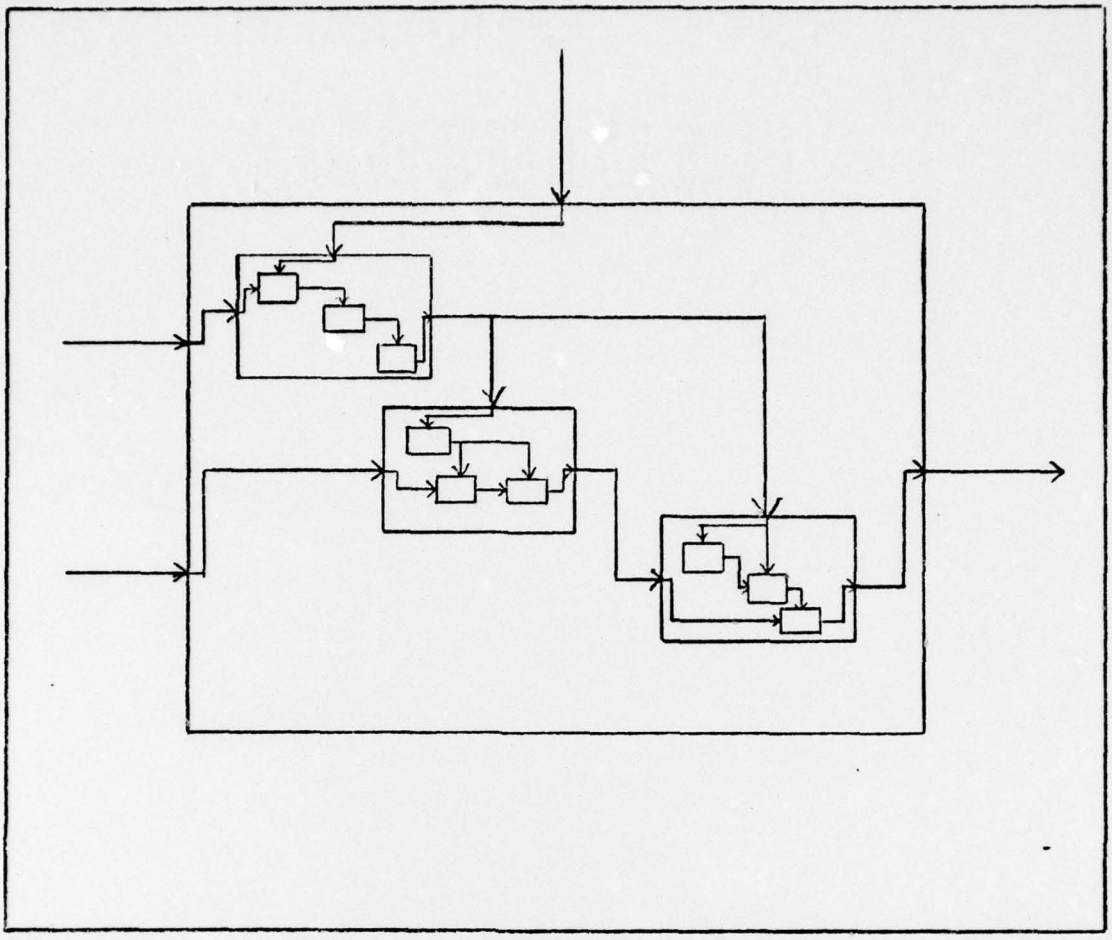


Figure A-1 Top-down View of an SA Model

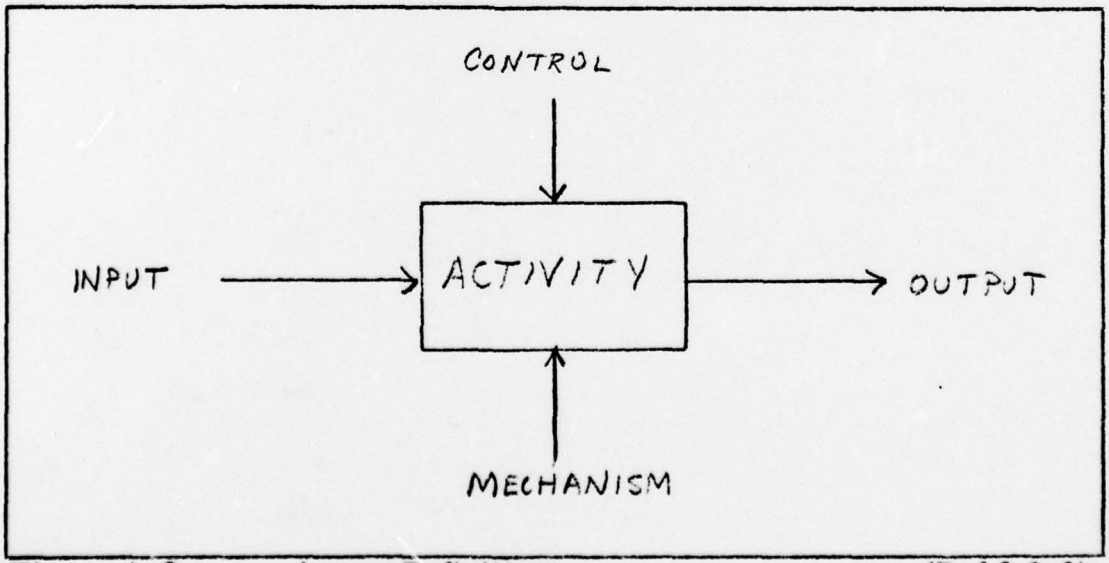


Figure A-2 Arrow Definitions (Ref 8:3-3)

control is data which may or may not be converted into output, but which in some way restricts the activity (starts or stops it for example). Every box must have at least one control arrow. A mechanism is a person or thing which acts as a processor. Mechanism arrows are often omitted when the processor is the same for all nodes. No limit is placed on the number of arrows which may interface with a side of a box, but it is common practice to group related types of data.

Between boxes, arrows may split and join. In general, all branches of an arrow contain the same data unless a branch is given a separate label. This convention is summarized in Figure A-3 which also gives two forms of OR-branches. The OR-branches are used to show that data follows one path or the other, but not both.

When two nodes are related so that the output of each is a control for the other, a special two-way arrow may be used. Figure A-4 shows a mutual control situation with a two-way arrow and the equivalent form with normal arrows. An arrow showing mutual control has two labels separated by a slash; the first label identified data going forward, and the second is the feedback data.

A special numbering system is used to distinguish between nodes at different levels and between nodes at the same level. In an activity model, node numbers are prefixed with the letter A. For preliminary nodes, A is followed by a dash and a number. Node A-1 may be used to show the model in relationship to other functions (Figure 2-2). Node A-0 serves as a cover sheet for the model; the node is simply a box

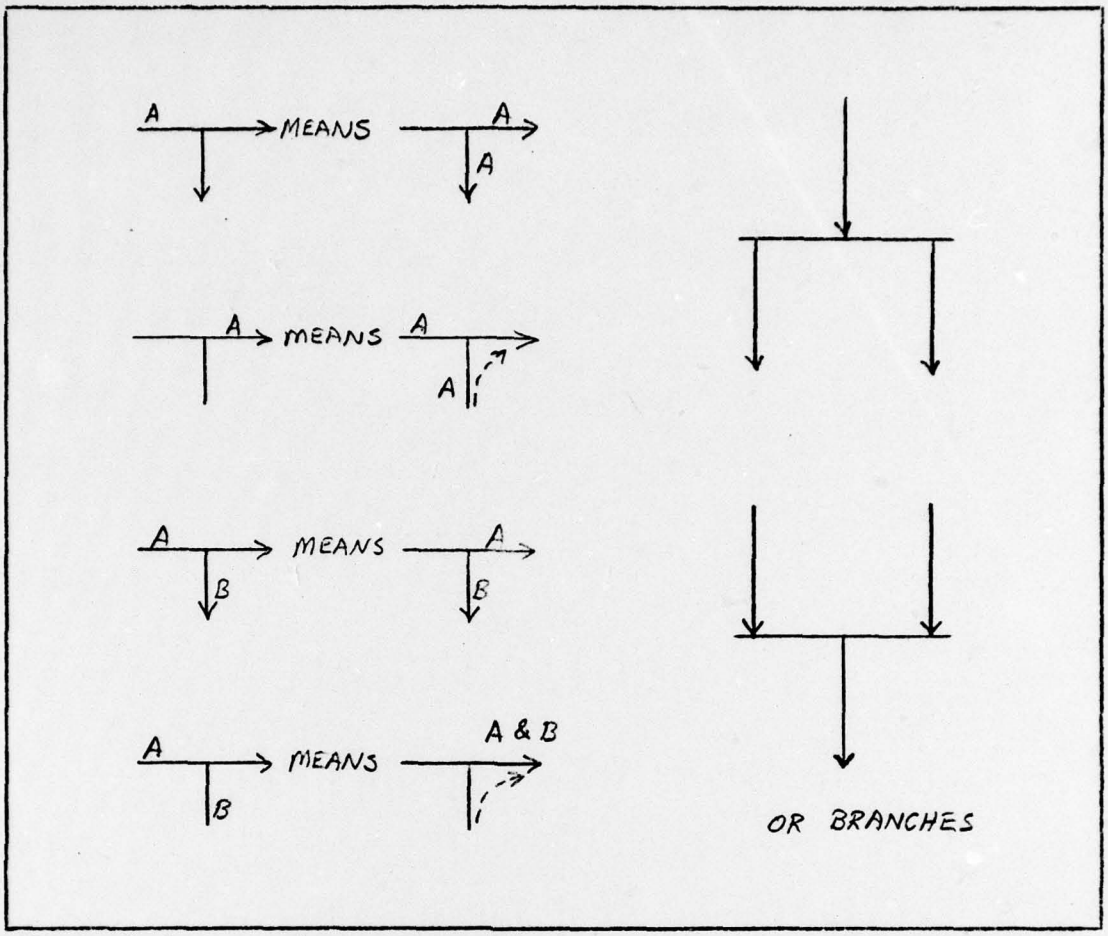


Figure A-3 Arrow Branches (Ref 8:3-13, 3-15)

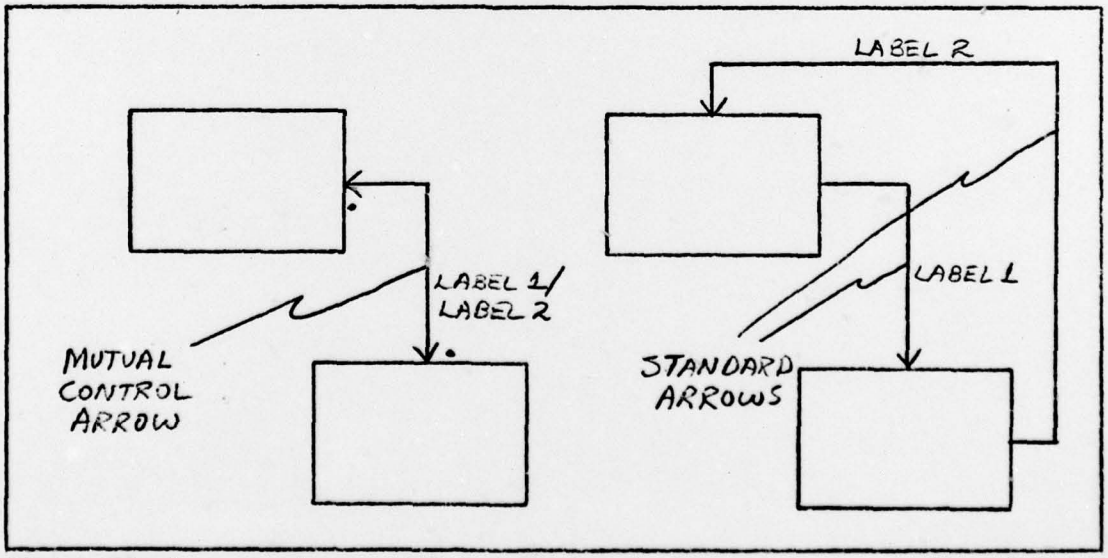


Figure A-4 Arrows Showing Mutual Control (Ref 8:3-16)

showing inputs, outputs, controls, and mechanisms for the function which the model is to describe (Figure 2-3). Decomposition begins in Node A0. Note in Figure 2-4 that each box of the decomposition is numbered; the boxes on all decomposition diagrams are numbered, and this number is used to form the node number. For the activities subordinate to Node A0, the node number is simply the box number on A0; Determine Net Count in Figure 2-4, for example, becomes Node A2. From this level on, the node number is a combination of the node number of the parent diagram and the box number of the subordinate. As an example, the decomposition of Determine Net Count is given in Figure 2-6. Box 4, Update Net Count, is assigned the node number A24. Subordinates of A24, if diagrammed, would have the numbers A241, A242, and so on through the last box number.

A special code called an ICOM code (Input, Control, Output, Mechanism) is used to identify arrows. The code contains a number, a letter, and another number. The first number is that of the box which the arrow enters or departs. The letter refers to the type of arrow, I for input, C for control, O for output, and M for mechanism. The last number distinguishes between arrows of the same type on a box; numbers are assigned counting from left to right and top to bottom. In Figure 2-6, the code 4I2 would refer to the arrow labeled State Change which is an input to Update Net Count, box 4. It should be apparent that an ICOM code is not unique because arrows can be connected to several boxes. Codes used in the text are derived from the box which

is of most interest in the discussion.

On a diagram, arrows which enter or depart the entire diagram are given a shortened version of the ICOM code. In the shortened version, the box number is omitted and the remainder of the code refers to the position of the same arrow on the parent diagram. For example, the output Net Count in Figure 2-6 is coded 01 because Net Count is the first output from box 2 of Node A0 in Figure 2-4 which is the parent diagram of Node A2.

A few important points about the text describing each SA diagram must be included here. The text is intended to point out the highlights of a diagram and not repeat all the details. As an aid to following the discussion, both the long and short versions of the ICOM code as well as box numbers are included in parenthesis following any reference to specific diagram features. Finally, as mentioned in Chapter II, the text describing the diagrams in this paper include timing and I/O specifications which is not a standard SA procedure.

VITA

John Robert Maneely was born on 10 April 1947 in Salem, Oregon. He graduated from Serra Catholic High School in Salem in 1965. He attended the University of Santa Clara and Oregon State University from which he received the degree of Bachelor of Science in Mathematics in June 1969. Following his graduation, he attended Officer Training School and received a commission in the USAF in March 1970. He completed pilot training at Williams AFB, Arizona, and received his wings in March 1971. For five years he served as an HC-130 rescue pilot, instructor, and flight examiner with the 41st Aerospace Rescue and Recovery Squadron at Hamilton AFB, California, and at McClellan AFB, California, and with the 56th Aerospace Rescue and Recovery Squadron at Korat Air Base, Thailand. He entered the School of Engineering, Air Force Institute of Technology in September 1976.

Permanent Address: 582 Wayne Dr. N.
Salem, Oregon 97303

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFIT/GCS/EE/78-4	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): DESIGN OF A LABORATORY DATA ACQUISITION SYSTEM (TIME DIGITIZATION SYSTEM).	5. TYPE OF REPORT & PERIOD COVERED: ~~MS Thesis~~ Master's thesis
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John R. Maneely [redacted]	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Project 7071-00-12
11. CONTROLLING OFFICE NAME AND ADDRESS: Guidance Test Division (GD) 6585th Test Group Holloman AFB, New Mexico 88330	12. REPORT DATE: Mar[redacted]78
	13. NUMBER OF PAGES: 212
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

JERRAL F. GUESS, Captain, USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Acquisition
Time Digitization
Microprocessor
Structured Analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A design was developed to show the feasibility of a special microprocessor based data acquisition system called a time digitization system which is to be used during tests of inertial guidance components for Air Force weapon systems. A digital system life cycle was developed to serve as a framework for the design project. Within the life cycle, the following phases were completed: system design, hardware selection/software structure, and circuit layout. A technique patterned after the Structured Analysis Design Technique was used to (over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

construct a requirements definition model. The requirements model was converted to a system design model by separating hardware and software functions. An Intel 8080A-1 microprocessor was selected to perform the software functions and MSI circuits were selected to perform specialized hardware functions. A software structure using finite state automata theory was created to control the data acquisition process.